Paolo Giorgini
Brian Henderson-Sellers
Michael Winikoff (Eds.)

LNAI 3030

Agent-Oriented Information Systems

**5th International Bi-Conference Workshop, AOIS 2003
Melbourne, Australia, July 2003
and Chicago, IL, USA, October 2003
Revised Selected Papers**

Springer

Lecture Notes in Artificial Intelligence 3030

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Paolo Giorgini Brian Henderson-Sellers
Michael Winikoff (Eds.)

Agent-Oriented Information Systems

5th International Bi-Conference Workshop, AOIS 2003
Melbourne, Australia, July 14, 2003
and Chicago, IL, USA, October 13, 2003
Revised Selected Papers

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Paolo Giorgini
University of Trento
Department of Information and Communication Technology
Via Sommarive, 14, 38050 Povo, Trento, Italy
E-mail: paolo.giorgini@dit.unitn.it

Brian Henderson-Sellers
University of Technology, Sydney
Faculty of Information Technology
Centre for Object Technology Applications and Research
PO Box 123, Broadway, NSW 2007, Australia
E-mail: brian@it.uts.edu.au

Michael Winikoff
RMIT University
School of Computer Science and Information Technology
GPO Box 2476V, Melbourne, 3001, Australia
E-mail: winikoff@cs.rmit.edu.au

Library of Congress Control Number: 2004106088

CR Subject Classification (1998): I.2.11, H.4, H.3, H.5.2-3, C.2.4

ISSN 0302-9743
ISBN 3-540-22127-1 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 11012078 06/3142 5 4 3 2 1 0

Foreword

This proceedings volume of the 5th AOIS Workshop is an opportunity for looking back at five years of organizing AOIS workshops. What did we achieve with the AOIS workshop series? Where were we five years ago, where are we now? Did our theme impact on the information systems field in the way that we had hoped for?

AOIS workshops have taken place in Seattle, Heidelberg, Stockholm, Austin, Montréal, Interlaken, Toronto, Bologna, Melbourne, and Chicago, always in conjunction with a major conference on either multiagent systems in artificial intelligence (AI/MAS) or information systems (IS). We have tried to innovate in holding these workshops as biconference events (each year AOIS held two workshop events, one at an AI/MAS conference and one at an IS conference), as well as using the AOIS web site as a medium for communication among researchers. So, certainly, we have reached a wide audience of researchers around the world from both the AI/MAS and IS communities. But did we also manage to build up a dedicated AOIS community?

Five years ago, we wrote: "Agent concepts could fundamentally alter the nature of information systems of the future, and how we build them, much like structured analysis, ER modeling, and Object-Orientation has precipitated fundamental changes in IS practice." Of course, a period of five years is too short for evaluating the success or failure of a new scientific paradigm. But still we may observe that while most IS conferences meanwhile list agents as one of their many preferred topics, agent-orientation is generally not considered to be a fundamental IS paradigm. Information systems are still viewed as data stores and not as software agents; customers are still viewed as "business objects" and not as (intentional) business agents an IS software agent has to interact with.

Why is this? Firstly, as always, there is resistance to change. Secondly, there are "cultural differences" between the AI/MAS and IS communities that hamper communication and slow down the exchange of new ideas and approaches. Thirdly, there are still not enough concrete success stories of agent-oriented techniques being exploited to bring benefits in real-world information systems applications. Finally, another obstacle is the need for standards before agent-oriented approaches can gain widespread adoption.

So, should we still believe what we wrote 5 years ago, i.e., that "Agent concepts, which originated in artificial intelligence but which have further developed and evolved in many areas of computing, hold great promise for responding to the new realities of information systems"? Yes! The limitations of today's IS technologies, in particular with respect to robustness, openness, flexibility, and ability to collaborate, call for further advances in IS concepts and techniques in the direction of agent-orientation. Further work in the area should clarify the benefits of agent-orientation and demonstrate the effectiveness of agent-oriented approaches to IS design. Efforts to develop effective techniques for the "semantic

web," which also exploit elements of agent-orientation, should also bear fruit for the IS area.

For this reason, we are looking forward to many more years of having the AOIS workshop as a bridge between IS and AI/MAS and as a catalyst for innovation in IS.

Eindhoven and Toronto, March 2004

Yves Lespérance
Gerd Wagner
Eric Yu

Preface

Information systems have become the backbone of all kinds of organizations today. In almost every sector – manufacturing, education, health care, government, and businesses large and small – information systems are relied upon for everyday work, communication, information gathering and decision-making. Yet the inflexibilities in current technologies and methods have also resulted in poor performance, incompatibilities and obstacles to change. As many organizations are reinventing themselves to meet the challenges of global competition and e-commerce, there is increasing pressure to develop and deploy new technologies that are flexible, robust and responsive to rapid and unexpected change.

Agent concepts hold great promise for responding to the new realities of information systems. They offer higher-level abstractions and mechanisms that address issues such as knowledge representation and reasoning, communication, coordination, cooperation among heterogeneous and autonomous parties, perception, commitments, goals, beliefs, intentions, etc., all of which need conceptual modelling. On the one hand, the concrete implementation of these concepts can lead to advanced functionalities, e.g., in inference-based query answering, transaction control, adaptive work flows, brokering and integration of disparate information sources, and automated communication processes. On the other hand, their rich representational capabilities allow more faithful and flexible treatments of complex organizational processes, leading to more effective requirements analysis and architectural/detailed design.

The Agent-Oriented Information Systems (AOIS) workshop series focusses on how agent concepts and techniques will contribute to meeting information systems needs today and tomorrow. To foster greater communication and interaction between the information systems and agents communities, the AOIS workshop is organized as a biconference event. It is intended to be a single "logical" event with two "physical" venues. This arrangement encourages greater participation from, and more exchange between, both communities.

AOIS 2003 was the fifth edition of the workshop. The first part was hosted on the 14th of July at AAMAS 2003 – the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems, in Melbourne, Australia, 14–18 July 2003. The second part was held in October at the 22nd International Conference on Conceptual Modeling, in Chicago (ER 2003). The workshop received in total 33 submissions, 16 of which were accepted for presentation (an acceptance rate of 48%). These papers were reviewed by at least 3 members of an international program committee composed of 34 researchers. The submissions followed a call for papers on all aspects of agent-oriented information systems and showed the range of results achieved in several areas, such as methodologies, applications, modelling, analysis and simulation.

This volume contains revised versions of 13 selected papers presented at the workshop. We believe that this carefully prepared volume will be of particular

value to all readers in these key topics, describing the most recent developments in the field of agent-oriented information systems.

We have grouped the papers into three categories: information systems and exemplar applications and case studies; methodologies issues; and, finally, modelling, analysis and simulation. Each section contains 5 papers selected from those presented at the two logical components of AOIS 2003.

The first group of papers discusses particular applications of agent-oriented information systems. Adam and Mandiam report on how they have created an MAS to address security issues in technological watch cells. Their approach, based on holonic as well as agent concepts, focusses on an integration of human (user) and organizational issues as well as agent-based information systems to support the organizational activities. They also describe their method (called AMOMCASYS) for building their MAS and evaluate their approach in a prototype implementation. In the second paper dealing with applications, Soh and colleagues describe an AOIS theory and prototype implementation for the support of teaching and learning. In their I-MINDS project, they show how real-time student response monitoring and feedback, performance evaluation and agent-supported creation of buddies groupings can be accomplished. This provides a more flexible and supportive environment in which students can learn more effectively and in which teachers can provide support more directly and efficiently. In the paper by Udupi and colleagues, we learn how agents have been used to help create trustworthiness in referral-based strategies for caching of information. By interacting with their neighbors, agents are found to develop their own trusted communities. In the next paper in this group, Rahwan and colleagues explore the applicability of agents to mobile computing. Using the AgentSpeak(L) language and the BDI architecture, they create an agent-based intelligent mobile assistant or AbIMA and test this on a handheld computer in situations where plans are volatile. The last paper in this group, by Wei and colleagues, applies agent technology to the problem of recommending information to the user. The particular context used is that of choosing appropriate secondary or sidebar information in a browser that will be of most use to the user. An auction mechanism is proposed and tested in a simulation environment. Evaluations are performed in terms of both user-perceived quality and also internal quality.

The second part of our book contains those papers that describe agent-oriented methodologies. We begin with a paper comparing existing AO methodologies. Here, Dam and Winikoff describe their evaluation of three AO methodologies (MaSE, Prometheus and Tropos) based on an assessment framework derived from the OO one of Ed Berard, covering the areas of concepts, modelling languages, processes and pragmatics. This helps them identify strengths and weaknesses and thus requires work to make these (and other) methodologies "commercial strength." In the second paper, Sturm and Shehory present a framework useful for the comparison of AO methodologies, as for instance described in the previous paper. They propose four major evaluation property groups: concepts and properties; notations and modelling techniques; processes; and pragmatics. They then construct a set of measures within each of these four

groups and, as an exemplar application of the framework, examine and evaluate the Gaia methodology. The next paper moves us to consider reuse in the requirements for an AOIS and stresses the importance of being purposive. Using a suite of four models derived from use cases, Goss and colleagues argue that these provide a good way of capturing the purpose of the information system, set in the business and human context. In the final paper in this methodology section, one specific AOIS methodology, MaSE, is examined in detail and weaknesses are identified. Using an iterative methodology improvement process, Vafadar and colleagues explain how they have extended MaSE by the addition of two steps and a new agent–object model to provide support for agents and objects within the same system.

The final section of the book contains papers describing agent-oriented modelling, analysis and simulation. The first of these demonstrates the value of an agent pattern language. The template they introduce is used as a vehicle for assessing whether an agent approach is both viable and useful. In the second paper, Bresciani and Donzelli investigate the requirements engineering support found in the Tropos AO methodology. They propose an agent-based Requirements Engineering Framework (REF) aimed at providing support for sociotechnical systems. They use a simplified version of the i* notation, illustrating their approach with a case study in e-government. Wagner then shows how the AOR modelling language can be used in discrete event simulation. Finally, using speech act theory, Bergholtz and colleagues present a unified framework to assist in integrating business models and process models, uniting the declarative nature of the former with the procedural and communicative aspects of the latter.

We thank the authors, the participants and the reviewers for making AOIS 2003 a high-quality scientific event.

March 2004

Paolo Giorgini
Brian Henderson-Sellers
Michael Winikoff

Organization

Organizing Committee

Paolo Giorgini (Co-chair)
Department of Information and Communication Technology,
University of Trento, Italy
Email: paolo.giorgini@dit.unitn.it

Brian Henderson-Sellers (Co-chair)
Faculty of Information Technology,
University of Technology, Sydney, Australia
Email: brian@it.uts.edu.au

Michael Winikoff (Co-chair)
School of Computer Science and Information Technology,
RMIT University, Melbourne, Australia
Email: winikoff@cs.rmit.edu.au

Steering Committee

Yves Lespérance
Department of Computer Science,
York University, Canada
Email: lesperan@cs.yorku.ca

Gerd Wagner
Department of Information and Technology,
Eindhoven University of Technology, The Netherlands
Email: G.Wagner@tm.tue.nl

Eric Yu
Faculty of Information Studies,
University of Toronto, Canada
Email: eric.yu@utoronto.ca

Program Committee

B. Blake (USA)
P. Bresciani (Italy)
H.-D. Burkhard (Germany)
L. Cernuzzi (Paraguay)
L. Cysneiros (Canada)
F. Dignum (The Netherlands)
B. Espinasse (France)
I.A. Ferguson (USA)

T. Finin (USA)
A. Gal (Israel)
U. Garimella (India)
A.K. Ghose (Australia)
B. Henderson-Sellers (Australia)
N. Jennings (UK)
G. Karakoulas (Canada)
K. Karlapalem (India)
L. Kendall (Australia)
D. Kinny (Australia)
S. Kirn (Germany)
M. Kolp (Belgium)
G. Lakemeyer (Germany)
Y. Lespérance (Canada)
D.E. O'Leary (USA)
F. Lin (Hong Kong)
J.P. Mueller (Germany)
J. Odell (USA)
O.F. Rana (UK)
M. Schroeder (UK)
N. Szirbik (The Netherlands)
C. Woo (Canada)
Y. Ye (USA)
B. Yu (USA)
F. Zambonelli (Italy)

Auxiliary Reviewers: Alexander Kozlenkov and Alexander Osherenko

Table of Contents

Information Systems and Applications

Methodologies

Modelling, Analysis and Simulation

Design of a MAS into a Human Organization: Application to an Information Multi-agent System

Emmanuel Adam and René Mandiau

LAMIH UMR CNRS 8530, University of Valenciennes
Le Mont Houy, 59313 Valenciennes Cedex 9, France
{emmanuel.adam,rene.mandiau}@univ-valenciennes.fr

Abstract. Web retrieval becomes more and more important for the knowledge management area, and we think that multiagent systems are a good answer to this problem. We propose, in this paper, a centralized Information multiagent system to help actors of technological watch cells (called CIMASTEWA). This system is an extension of a previous distributed multiagent system and is set-up within n-tier architecture. In order to make the search secure, notably concerning surveillance by spies, we have added particular search strategies. Furthermore, in order to encourage the users to share their results, we propose to organize groups of users according to their centre of interests. This functionality is one of our main perspectives concerning this information multiagent system that has been developed in answer to demands from technological watch cells.

1 Introduction

The boom in Internet technology and company networks has contributed towards completely changing a good number of habits which have been well-established in companies for several decades. In a context of globalisation of the economy and of serious modifications in socio-economic structures, the technical and administrative processes which underlie the activities of a Company are, in particular, the subject of considerable revision. Documents on paper, exchanged from hand to hand are progressively being replaced by electronic documents transmitted automatically by machines without taking the human factors, such as the notion of the group (the individuals are isolated at their work post), the levels of responsibility or even man-machine co-operation into account. Admittedly, tools for aid in co-operative work have already been suggested, some with success, but they do not tackle the overall organisation.

This fact formed the base problem of our work. Indeed, our research aims to set up an information management assistance system in watch cells or in laboratories. In order to take into account the human factors, such as the notion of the group or even the man-machine co-operation, we have previously developed a method (AMOMCASYS, meaning the Adaptable Modelling Method for Complex Administrative Systems) to design and to set-up multi-agent systems within human organization, more precisely in the cooperative processes of these organizations [1]. We have reused this method to develop multi-agent systems for helping cooperative information management within a team of technological watch. The main advantage of our method, and of our system, is that it takes into account the cooperation between actors

P. Giorgini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 1–15, 2004.

of workflow processes. Indeed, we have noticed that most human organizations follow a holonic model (each part of the organization is stable, autonomous and cooperative and is composed of sub holonic organizations to which it is responsible) and we have built our method by integrating these notions.

This paper describes firstly the AMOMCASYS method that we have built to model human processes and to design multi-agent systems, which interact with actors of the processes. Then, the method that we propose for design of such multi agent systems is shown in two steps: a step for the individual characterization of the agents; and a step for the design of the agents' cooperative working. Finally, this article presents an application of our method to the design of multi-agent systems that help actors of a technological watch cells to search and exchange information. This third-part application is an extension of a previous distributed version [2].

2 Human Organisation Modelling

2.1 Holonic Principles

Before designing a multi-agent system to assist management of information within a human organization, we think that it is necessary to understand its working mechanisms. This should allow us to take into account all of its characteristics and to introduce multi-agent systems in a relevant way.

We have shown in [1] that complex administrative systems follow the holonic model proposed by Arthur Koestler in 1969 [3].

A holon, which is a part of a holonic system, is defined by Koestler as being a part of a whole or of a larger organization, rigorously meeting three conditions: to be stable, to be autonomous and to be co-operative[1]:

- stability[2] means that a holon is able to cope and react when it is subjected to high demand or to major disturbances,
- autonomy suggests that a holon is capable of managing itself when subjected to demands in order to achieve its own aims,
- the capacity to co-operate means that the holons are able to coexist with other holons or other layers of holons, and are capable of working on common aims and projects.

Here we can find at least two of the characteristics of the agents in a MAS sense: autonomy and co-operation [4]. The third characteristic, the capacity to adapt itself to an environment, is suggested by stability. Holons of holonic system are organized following levels of responsibility. So, a holonic multi-agent system has a hierarchical structure where each agent is responsible for a sub-holonic multi-agent system. This notion of recursivity is very interesting to designing such systems. Actually, each part of the holonic system is designed globally as the global system (with the addition of specialities).

---

[1] A. Koestler proposed 65 rules describing the architecture of a holonic system and the communication rules between entities of the system.

[2] However, if the holons are stable, they do not have to be rigid. Indeed, the stability of the whole system is more important than the stability of each of its parts. So, it is sometimes necessary that some holons be temporarily destabilized so that the whole system can take more long-term protection strategies.

As the holonic architecture is a well adapted or human organization where actors exchange between them some information, we have proposed to reuse this architecture to design multi-agent system that have to manage and exchange data.

Our aim is the design of a multi-agent organization providing assistance to the actors in cooperative processes. This organization must be fixed (which does not imply rigidity) in order to be able to meet the user demands as quickly as possible. This is why we have used the social rules defined in the holonic concept in order to simplify and to accelerate the design of a multi-agent society (in the [5] sense). This holonic concept is especially useable and useful in structured and cooperative fields [6].

Indeed, the hierarchical structure is not sufficient to model modern organisations and bureaucracies: "the degradation of hierarchy is a necessity for organisation to prosper" [7]; the organisations are, or have to be, flexible, more decentralised, based on roles.

We find these requirements in the area of the intelligent manufacturing system, where new concepts have been proposed such as:

- the bionic manufacturing systems (BMS) [8], where the organisation is seen as an organ composed of cells (called modelon), which can be merged or divided into units,
- the holonic manufacturing systems [9](HMS), based on the holonic concept,
- and more recently the fractal manufacturing systems (FrMS) [10], which give a hierarchical view of the organisation where each basic part (BFU for Basic Fractal Unit) owns an individual goals and offers services. The global coherence of the system comes from inheritance mechanisms, particularly for the goal formation process.

All of these concepts allow to decompose recursively an organisation into entities which can be themselves decomposed into other entities. Each of these entities is autonomous, flexible and cooperates with other ones to maintain the organisation's stability. The main differences between these approaches concern [10] the definition of entities groups, which are predefined in the holonic concept and in the fractal concept (with dynamic redefinition) and dynamic in the bionic concept; the dynamic reconfiguration of the system that could imply the change of dataflow in the bionic and fractal concept and that consists of evolutions of predefined strategies (canons variations) in the holonic concept.

We think that the fractal and bionic approaches are too permissive, regarding the reconfiguration capacities, for the design of stable software. Moreover, holonic concepts are closer to human organisations in which we want to integrate software agent systems.

However, before setting-up a multi agent system, and more generally software, it is necessary to model the human organization in which it has to work. That is why we have proposed a method adapted to human organization where actors have different levels of responsibility and where actors have to cooperate around documents.

2.2 An Adaptable Modelling Method for Complex Administrative Systems (AMOMCASYS)

The particular architecture of complex administrative systems has led us to choose a method able to take into account all of their specificities, with an aim of designing

tools supporting cooperative work. That is why we have designed a framework of comparison [11]. This framework, based on [12], is composed of 76 criteria merged in 5 dimensions related to: the used formalisms; the type of organization and environment tackled; the underlying methodology; the aimed tools; and the integration of communications and relations between actors. This framework has been extended by [13] to compare multiagent system modelling methods.

The comparison has shown that none of the eight compared methods, which are usually used in the domain of the software engineering, fully answers our criteria. Hence, a modelling method (AMOMCASYS for Adaptable MOdelling Method for Complex Administrative SYStem), adaptable according to the system to study, has been set-up, on the basis of the more pertinent parts of these methods.

Regarding the system's modelling, the selection criteria, for the comparison as well as for the proposition of AMOMCASYS, are: the clarity (primordial for the confrontation of the models with actors of the system); the representation of the communicated objects (the documents), the data flows, the cooperation between actors, and the responsibility levels. So, AMOMCASYS uses four models: a data model (the data model of UML)[14], a data flow model (adapted actigrams of the SADT method[15]), which allows representing responsibility levels), a data processing model (the processing model of CISAD method[16]), and a dynamic model (parameterized Petri nets[17]). Each of these models brings a complementary view on the modelled system. They are not all necessary: in some cases, it is possible to use only a data flow model or a data processing model. It is in this sense that AMOMCASYS is an adaptable method, but in all cases it is indispensable for building the data model, representing all the data, or the most important types of data, used during groupware processes.

Before setting up groupware software, AMOMCASYS proposes to actors of the complex administrative processes to simplify them (that is to say decrease the number of communications, of database duplications, data controls…). It has then been necessary to provide processes' managers (department managers and/or quality managers) with an easy-to-use CASE Tool (Visual Basic layer based on the commercial software VISIO), allowing them to model the processes.

Three steps are necessary to set-up a MAS with AMOMCASYS (cf. Figure 1):

- firstly, an analysis has to be done in the department;
- next, the processes where the multi-agent has to be set-up are modelled by using the data model and the dataflow model (and some times the data processing model). Actors are shown the models in order to involve them in the setup of the project and in order to validate the model. Some organisational optimization of the processes are sometimes made at this stage[3];
- finally, the data exchanges and the working mechanism of the multi-agent system are modelled with the processing model and the data model is used to represent classes that compose the multi-agent system. A simulation of the human activities can be used to determine the opportunity to use agents on particular points of the process.

[3] For example, the time for dealing with one procedure involving about 15 actors was halved, by improving cooperation and increasing the responsibilities of the actors.

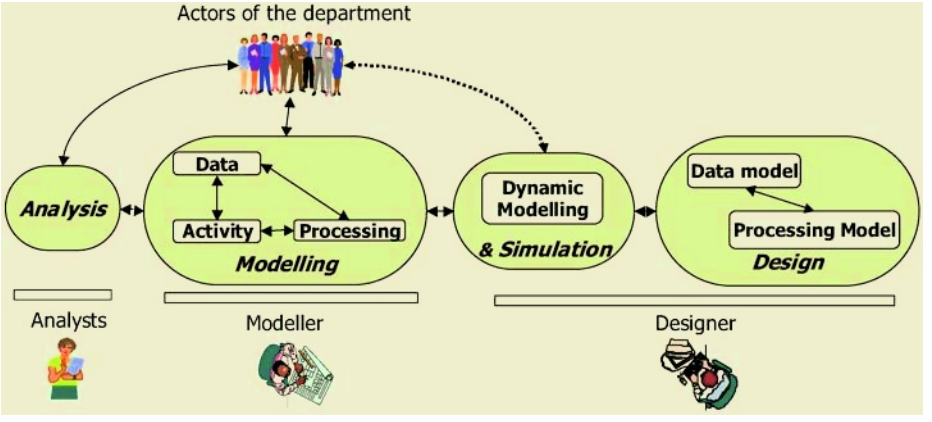

Fig. 1. Steps of the AMOMCASYS method.

3 Individual and Collective Activities of Agents into a MAS

Although the definition of our MAS structure has been facilitated by the use of holonic principles, the modelling of the system organization and the characterization of the functionality of the agents remain problematic. Indeed, the research published on this subject is mainly theoretical. Only a few recent research projects allow organization modelling with an application goal, but they are mainly devoted to the field of collective robotics [18] [19].

Here, we propose a specification in two stages: the first stage concerns the individual functioning of each type of holonic agent; the second concerns the functioning of the group, describing communications between agents and between actors and agents.

3.1 Individual Activities of the Holonic Agents

In order to describe the general characteristics of various types of agent, we use a grid adapted from Ferber [20]. This grid enables us to define the functions for each holonic agent relating to: knowledge (the representational function also describes the non-procedural knowledge); action planning (the organizational function); interactions; the maintenance (the preservation function) and the actions specific to the role of the agent (the productive function). These functions are described in three dimensions: according to the agent's environment, to the other agents and to the agent itself.

This grid is applied to the design of the different roles of agents. It is also applied to the definition, at a higher abstraction level, of the different functions of the multi-agent system (because, in our case, a MAS can be considered as a single holonic agent).

The grid of Table 1 enables us to have a clear view of the agents' actions according to the environment and according to the other agents. Now, we need to define the functioning of the whole organization, of the cooperative functioning of the agents. For that, it is necessary to use a method, such as AMOMCASYS.

Table 1. Design grid adapted from Ferber's analysis grid [20].

Dimensions \ Function	*Social*	*Environmental*	*Personal*
Representational	Representation of the group (of the other roles)	Representation of the world	Representation of itself, of its capacities
Organizational	Planning of social actions, communications	Planning of actions in the environment	Planning control, meta-planning
Interactive	Description agent-society interaction, performative	Perception and action mechanisms in relation to the environment	Auto-communication Auto-action
Productive	Management, coordination and negotiation tasks	Analysis, modification and creation tasks	Auto-modification, learning
Preservation	Preservation of the society, the relations, the network of contacts	Preservation of resources, defence and maintenance of territory	Self-preservation, repair, maintenance

3.2 Cooperative Working of the Holonic Agents

Regarding the design of the MAS to be integrated to the human process, AMOMCASYS data model allows us to represent the principal holon class (which describes the general agent structure) as well as the classes associated with the knowledge of the environment (representation of the process, the actor, the workstation, the responsible, the subordinates). Each holonic agent has five main functions: to plan its action according to the process and its current state (corresponding to the organizational function); to receive and send to other holonic agents (corresponding to the interaction function); to act (corresponding to the productive function, to the specialty of the agent) and to manage the links between the responsible and the subordinates (corresponding to the preservation function). Of course, each holonic agent has an implicit function: 'initialize' (enabling it to acquire knowledge upon the MAS).

The four main functions (the organizational, interactive, productive and conservative functions) imply co-operations between holonic agents and sometimes between the agents and the actors (the users). The processing model of the AMOMCASYS method can model these co-operations, as we will see in the following case study.

4 Design of a MAS into Technological Watch Departments

The case study that is presented in this article refers to a technological watch department of a large company. In this application, we have designed a MAS in order to assist actors of a technological watch department [21] in their tasks. This specification has been undertaken following the analysis and the modelling of the department's processes. In these processes, actors (called watchmen) have to collect information on the Internet, to manage them and to distribute them to their clients. So we had to design a MAS for the information retrieval. The MAS (CIASCOTEWA meaning COoperative Information Agents' System for the COoperative TEchnological WAtch) that we have proposed to the actors was in fact composed of a sub-MAS (CIASTEWA, for CO-operative Information Agents' System for the TEchnological WAtch), which is dedicated to a single user [2].

Each CIASTEWA (Figure 2) is typically an information multi-agents system (IMAS), being composed of:

- Information agents that search, on the basis of requests that are sent to them (directly by an interface agent or indirectly through of a database), information on databases (local or distributed) or on Internet sites. They are called, in our case, search engine agents.
- Several layers of coordinator agents: the first layer, composed of request agents, coordinates activities of the search engine agent; the second layer, composed of one information responsible agent that coordinates the activities of the request agents; the last layer is composed of the coordinator agent, which coordinates activities of the information responsible agent and of the interface agent. All of these agents own knowledge on the agents for which they have responsibility (such as their addresses, their search domains for example).
- One interface agent, that assists the user to express their requests and allows them the interaction with the results provided by information agents, or with the other users of the group.

The knowledge of the CIASTEWA on the user (its name, its requests, their results) are stored in a local database.

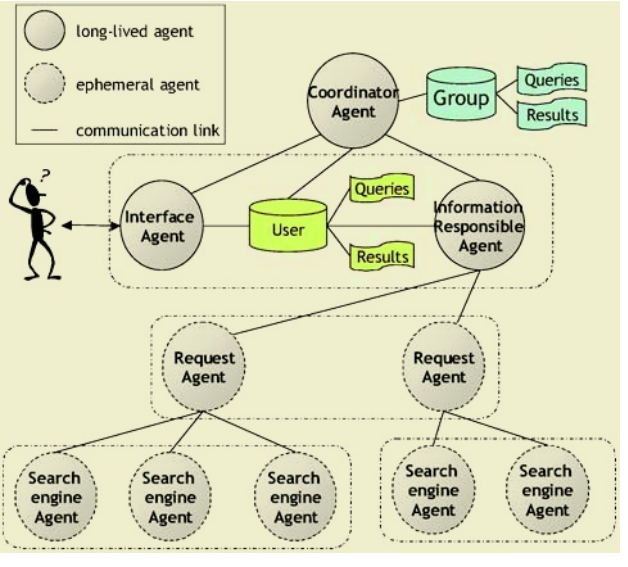

Fig. 2. Architecture of a CIASTEWA.

When a CIASTEWA performs a search, it asks its acquaintances (the other CIASTEWA) if they do not own the same results that it just get from Internet. Each user is so informed if another user received the same information as himself/herself. We think that this should encourage actors to cooperate.

The role of a CIASTEWA is not to only help the user, to whom it is dedicated, to search relevant information, but also to help him/her to communicate it with other

actors. In order to maintain or create the feelings of a community or group among the actors, which is often forgotten with the use of new technologies (the individuals are isolated with their workstation), we have developed self-organizing capacities, based on the Kohonen algorithm, in order to generate communities of CIASTEWA, which have to respond to the same kinds of requests. This reorganization is indicated to users in order to encourage them to cooperate, if they want to do it, with other users having the same centres of interests.

However, although that the system proposed is flexible (it is easy to add or remove a user from the system) and cooperative, the users suggested the fact that some data are duplicated on their own computers, and are not necessarily protected against hackers (all the users do not use a personal firewall). So, in order to avoid redundancies of data, to securitize them, we propose to use a centralised database and to put a CIASTEWA at the end of a third-party application (cf. Figure 3). We call this system a CIMASTEWA (for Centralized IMAS for TEchnological WAtch).

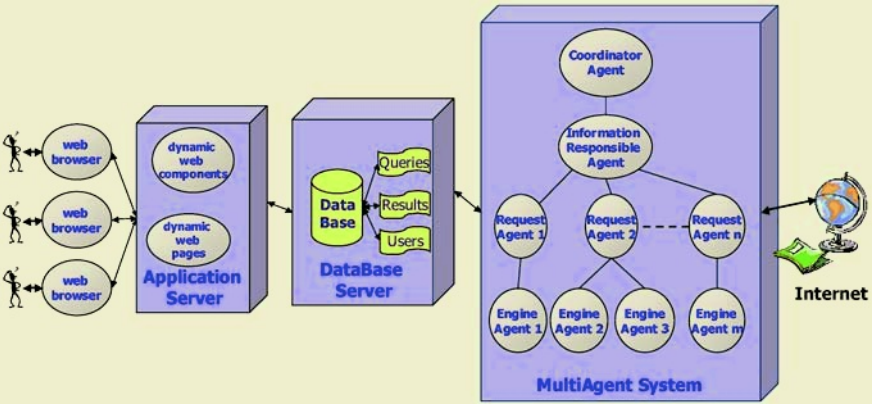

Fig. 3. Architecture of the CIMASTEWA.

We removed the interface agent from the CIMASTEWA. The users have access to this IMAS through their browsers by calling dynamic web pages (jsp or asp pages for example) and dynamic web components (like java bean for instance). They record their queries in the centralized database that the coordinator agent checks periodically to detect if a request has to be made. In this case, the request is sent to the information responsible agent, which distributes it to the information agents according to a search strategy. The information agents send back their results to the information responsible agent, which filters and merges them to record them in the database.

Our MAS is not centralized on a single computer but works on a network of computers. So it is possible to have more sophisticated principles of search. For example, information search, in the technological watch domain, implies the development of securitized search strategies, such as adding 'noise' around the requests. So, in the CIMASTEWA, if a user wishes one of its requests to be scrambled, this one is then decomposed into elementary requests, which are launched with false requests. The real results are then recomposed by the information responsible agent with a particular process.

Regarding the notion of cooperation, we propose to the user to share some of their requests by defining them as public, so that other users have access to the public requests and to their results.

In order to design the CIMASTEWA, we have reused the methodology described in this article, and that we have applied in previous projects: firstly, the individual roles are described; secondly, the cooperative activities of the agents are designed.

4.1 Individual Design of the CIMASTEWA Agents

Four roles have to be designed in a CIMASTEWA: the coordinator agent role ( ), the information responsible agent role ( ), the request agent role ( ), and the search engine agent role ( ).

Table 2 presents the behaviour of the coordinator agent. The roles of the other types of agents are described in the same way.

Table 2. Definition of a coordinator agent.

Dimensions/ Function	*Social*	*Environmental*	*Personal*
Representational	Knows the and	Knows the requests and results of the users.	Knows its name, its IP address.
Organizational	Watches over the coordination of actions of the group of agents of the CIMASTEWA.	Manages the database.	Checks that a request does not exist
Interactive	Is the responsible of .	Interacts with the database and the jsp pages	Ø
Productive	Send requests to the appropriated	Fills the database with results and requests provided by the	Makes modifications of knowledge, of its role.
Preservation	Checks if its contacts are active…	Check the database.	Deferred to its contacts.

4.2 Cooperative Functioning of a CIMASTEWA

After having described the individual roles of agents that compose a CIMASTEWA, we have to define the cooperative interactions between them. An extract of the processing model representing a cooperative activity within a CIMASTEWA is shown on Figure 4.

This figure presents the recording of a new request in a CIMASTEWA. The user adds a request in the system through the jsp pages and decides to launch the request immediately. A message is sent to the coordinator agent which informs the user if the request exists in the group database, if it is a subset of another one or if it includes requests of other actors. The request is then recorded in the user database and a message is sent to the information responsible agent. This message asks it to execute the

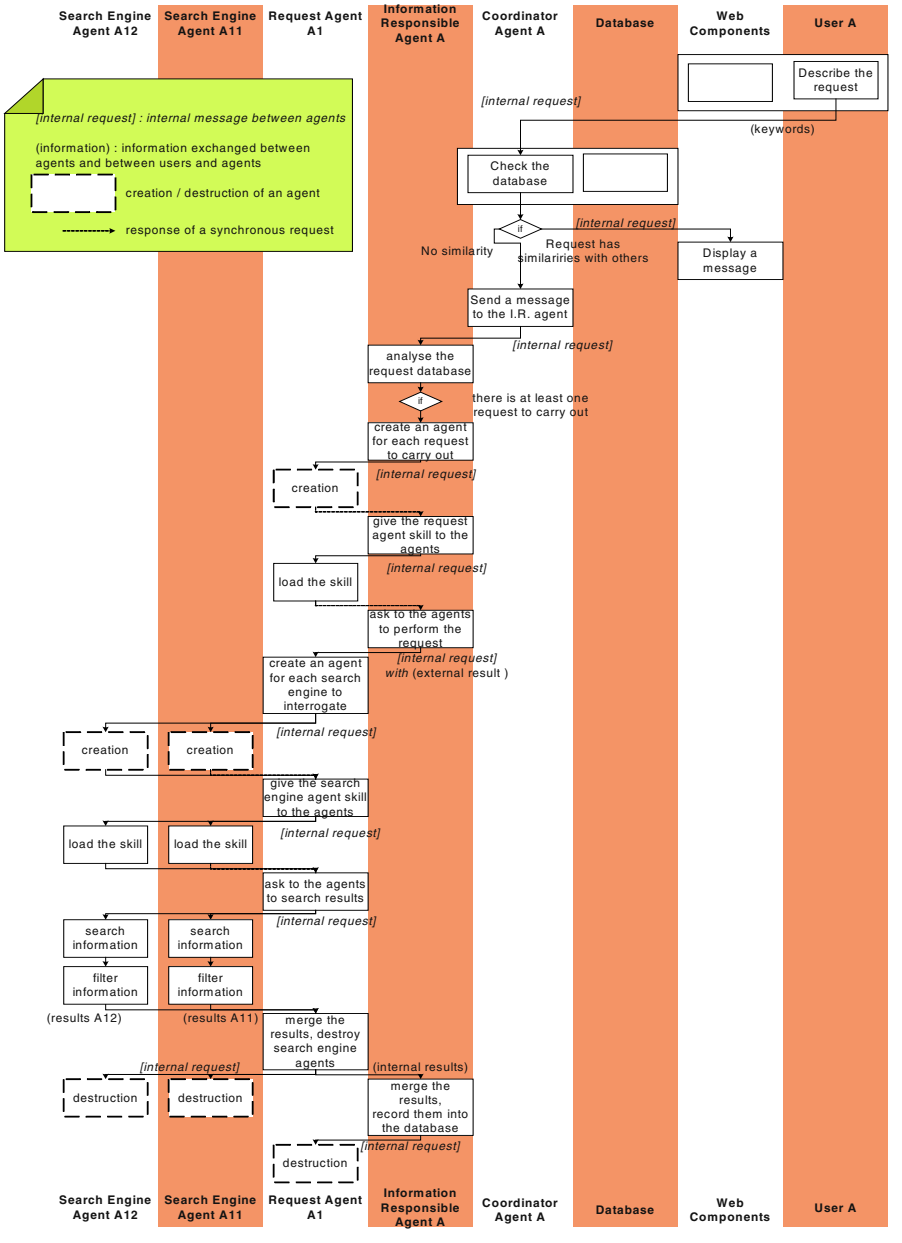

Fig. 4. Processing model of the request recording process in a CIASTEWA.

requests not yet carried out. For this, the agent creates a request agent for each request. Each request agent creates a search engine agent for each search engine specified in the request. Each of these request agents connects to the Internet in order to find results and send them to its responsible. When the request agent has received the

results of each of its subordinates, it filters them (it deletes the doubles) and sends the results to its responsible. When the information responsible agent has received a response from each of the request agents, it records them in the database.

Thanks to the AMOMCASYS method, we have defined other interactions between users and agents of the CIMASTEWA such as the annotation, the deletion of a result, the modifying and the deletion of a request.

4.3 Application of the CIASCOTEWA

The CIMASTEWA has been developed, mainly in Java, in a small data-processing company to answer the demands of technological watch cells and is currently used in our laboratory. The coordinator agent, the information responsible agent and the information agents have been developed in Java, using the MAGIQUE [22] multiagent platform, which is particularly well adapted to the hierarchical structure of our MAS. Currently, the search strategy used by our information agent is a broadcast strategy: the information responsible agent sends the request that has to be launched to information agents, which are each dedicated to a search engine. So our system is able to search information on 5 classical search engines and two search engines that are specialized in magazines and documents.

The following code presents an example of creation of the multiagent system composed initially of the three main agents: “Coordinator”, “Interface” and “Information-Responsible”. MAGIQUE agents are empty shells having only communication capacities. It is possible to give skills to an agent by associating it with java classes. Messages exchanged between agents, which are called requests, consist of calls to functions or sub-processes that are located in the skills[4].

Extract of the program that create the MAS (extract of the main method of the CIASTEWA class).

```
// agents creation
Agent sup     =  createAgent("Coordinator");
Agent display =  createAgent("Interface");
Agent search  =  createAgent("InformationResponsible");
// agents connection
display.connectToBoss("Coordinator");
search.connectToBoss("Coordinator");
// skills addition
sup.addSkill(new CoordinatorSkill(sup, configFile));
display.addSkill(new InterfaceSkill(display));
search.addSkill(new
                  InformationResponsibleSkill(search));
/* 'agentInitialize' method of the Coordinator agent
asked */
sup.perform("agentInitialize");
/* Coordinator agent asks to the InformationResponsible
agent to launch the go method */
sup.perform("InformationResponsible", "go");
```

[4] If a request cannot be satisfied, because the agent does not know how to answer (the function asked is not present in its skills), it is stored by the platform until the agent learns to answer to it.

Extract of the 'go()' method of the InformationResponsibleSkill class, which is associated to the Information Responsible agent. This code creates an Request Agent for each search asked by the users.

```
for (int i=0; i<reqToLaunch.size(); i++)
{
  Search req = (Search) reqToLaunch.elementAt(i);
  /* creation of an empty agent whose the name is the
  name of the request */
  Agent a = platform.createAgent(req.getName());
  // addition of the skill
  a.addSkill(new RequestSearchSkill(a, req));
  /* connection : Information Responsible agent is the
  boss of the Request agent */
  a.connectToBoss(getName()+"@"+IPADDR+":"+PORT);
  /* asking for the execution of the searchRequest
  method of the request agent */
  perform(req.getNom(), "searchRequest");
  nbRequestLaunched ++;
}
```

In these code examples, agents are created locally, but their creation on remote machines is similar.

The main advantage of using agents, rather than an application based only on threads, is based on the flexibility offered by this technology. Indeed, for example, each agent can be situated on a remote machine or can move on a new computer (only if it is not acting) in order to beneficiate of more resources. Likewise, an agent can dynamically learn new skills or change its skills to adapt and increase its capacity of search (for example, it is possible to add a new skill dedicated to the information pertinence evaluation, that is not taken into account currently).

The application server that we use is Tomcat, three databases have been developed (in Progress®, in Ms Access® and in MySQL, which is the version actually used). The MAS has no particular location; it only needs to have access to the database and to the Internet, in order to execute the searches.

The human machine interfaces have been developed in web pages (using jsp, javascript and css processing language). Through these pages, the users can: add, modify, remove requests; consult previous results; choose to share or not their own requests or results; separate a request into two parts; merge results of two requests (in fact, this duplication is the first strategy of scrambling).

Figure 5 presents a screen copy of the web page allowing the user to consult personal or public results: to send them to another actor (through the user mail service), to add some information, to archive them and/or to delete them. The results proposed by the CIMASTEWA of Figure 4 come from a request on 'collaborative search engine'.

The functionality concerning the identification of interest communities of the users has not been yet added to this new IMAS. Currently, we are developing the functionality of self-organizing in the CIMASTEWA, which will allow it to choose the best search strategy according to the users or to the requests.

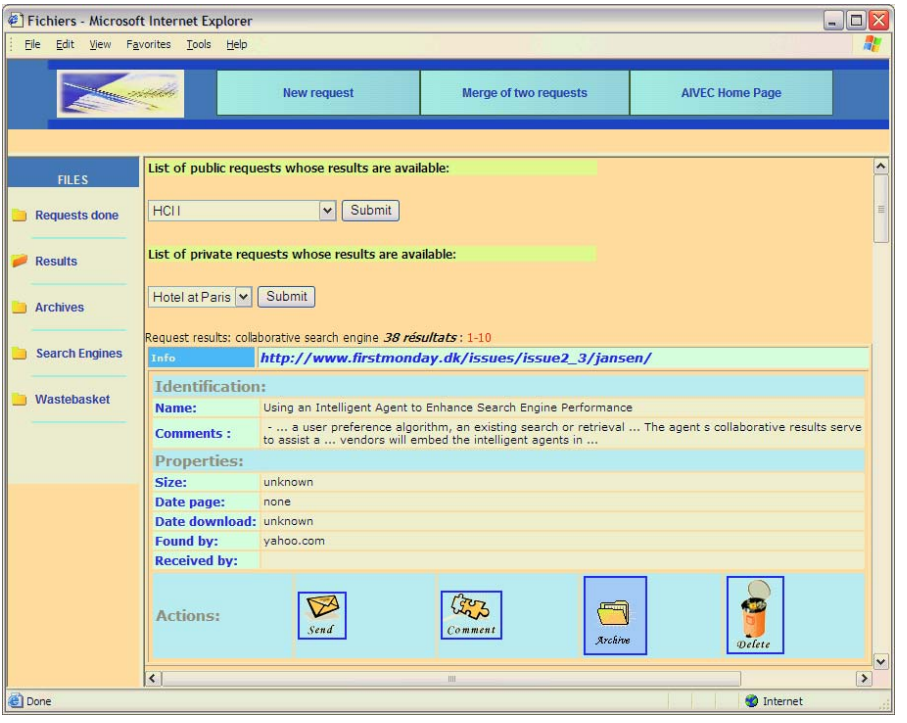

Fig. 5. A screen copy of a CIMASTEWA.

As perspectives, according to the user needs, it could be interesting to have different search strategies for the Search Engine Agents. For example: if the user wants to have results rapidly, an invitation to tender can be launched to these agents; if the user wants to have all the possible results, then the requests are sent to all of them by a broadcast technique. On the other hand, according to the queries, different search strategies can be also chosen. For example: if the words used in the request correspond to specialties of information agents, then the search is organized by specialty; if some requests are subsets of other requests, then information agents are structured hierarchically.

We are currently working on the "regeneration" of some parts of the CIMASTEWA that could be broken as a result of a technical problem such as the crash of a computer or a search engine malfunctioning. This will make our MAS more flexible and so will more justify the use of this kind of technology.

We think that the main interest of our system is the personalisation of the search strategies that it proposes. Indeed, it is relatively easy to add skills to the agents, or to new kinds of agents, in response to particular needs (in case of the addition of a new kind of agents, their roles and their relations with the other agents have to be designed according to our approach).

5 Conclusion

In order to specify a co-operative information agents' system into a human organization, we have defined a method composed of three steps: a step of analysis and modelling of the human organization; a step of modelling the insertion of agent systems into the human organization; and a step of design of the multi-agent system.

Our work uses the AMOMCASYS method that we have defined to analyze and model complex administrative system, underpinned by a holonic model. Indeed, this model allowed us to understand human organizations and we have showed that this model supports the design of a multi-agent system particularly adapted to the studied human organizations.

We have developed a Centralized IMAS to help actors of technological watch cells, or searchers in a laboratory, that have to search information, to share it in a cooperative aim. Our system proposes a search strategy in order to hide the actual searches asked by the users. Sharing the queries and their results allows the user to keep in mind the notion of a group, which is important, and allows them to win time in their searches. This system is currently used in our laboratory by a few searchers of our team. We plan to set-up this system in a larger way in our laboratory in the short term. Indeed, we are carrying out research and development: to self-organize the information agents in order to have more relevant results; and to identify communities of users automatically. In this way, we could measure the impact of our MAS on the behaviour of the searchers.

References

1. Adam, E., Modèle d'organisation multi-agent pour l'aide au travail coopératif dans les processus d'entreprise : application aux systèmes administratifs complexes (in french), PhD Thesis, Université de Valenciennes et du Hainaut-Cambrésis, (2000).
2. Adam, E., Mandiau, R. Bringing multi-agent systems into human organizations: application to a multi-agent information system. In M.A. Jeusfeld, O. Pastor (Ed.), Conceptual modeling for novel application domains, ER 2003 Worshops ECOMO, IWCMQ, AOIS, and XSDM, Chicago,IL, USA, October 2003 Proceedings, LNCS 2814. Berlin: Springer (2003) 168-179
3. Koestler, A. The Ghost in the Machine. Arkana Books, London, (1969).
4. Nwana, H. S. Software Agent: an overview. Knowledge Engineering Review, 11(3) (1996) 205-244.
5. Mandiau, R., Le Strugeon E. & Agimont G. Study of the influence of organizational structure on the efficiency of a multi-agent system. Networking and Information Systems Journal, 2(2) (1999) 153-179.
6. Gerber, C., Siekmann, J., Vierke, G. Holonic Multi-Agent Systems. Research report, RR-99-03, March, DFKI GmbH, Germany, (1999).
7. Schwarz, G.M., Organizational hierarchy adaptation and information technology, Information and Organization 12 (2002) 153–182
8. Okino, N., Bionic Manufacturing System, in J. Peklenik (Ed.) CIRP, Flexible Manufacturing Systems: Past-Present-Future (1993) 73-95.
9. Van Brussel, H., Bongaerts, L., Wyns, J., Valckenaers, P., Van Ginderachter, T., A Conceptual Framework for Holonic Manufacturing Systems: Identification of Manufacturing Holons. Journal of Manufacturing Systems, 18, 1 (1999) 35-52.

10. Kwangyeol, R. and Mooyoung, J., Agent-based fractal architecture and modelling for developing distributed manufacturing systems, Int'l J. of Production Research, Vol. 41, No. 17 (2003) 4233-4255.
11. Adam, E., Kolski, C. Etude comparative de méthodes de génie logiciel utiles au développement de systèmes interactifs dans les processus administratifs complexes. Génie Logiciel, 49 (1999) 40-54.
12. Pascot, D., Bernardas, C. L'Essence des Méthodes : Etude Comparative de Six Méthodes de Conception de Systèmes d'Information Informatisés. Proceedings INFORSID'93 «Systèmes d'information, systèmes à base de connaissances», Lille, France, (1993).
13. Sabas, A., Badri, M. & Delisle, S. Applying a New Multidimentional Framework to the Evaluation of Multiagent System Methodologies. International Symposium on Information Systems and Engineering (ISE 2002), San Diego (California, USA), Simulation Series, vol. 34, n° 2 (2002) 37-44.
14. Unified Modeling Language (UML), version 1.5, (formal/03-03-01) (2001) OMG
15. I.G.L. Technology. SADT, un langage pour communiquer. Eyrolles, Paris, (1989).
16. Dumas, P., Charbonnel G. La méthode OSSAD, pour maîtriser les technologies de l'information. Tome 1 : principes. Les éditions d'organisation, Paris, (1990).
17. Gracanion, D., Srinivasan, P., Valavanis, K.P. Parameterized Petri nets and their application to planning and coordination in intelligent systems. IEEE Transactions on Systems, Man and Cybernetics, 24 (1994) 1483-1497.
18. Collinot, A., Drogoul, A. Approche orientée agent pour la conception d'organisations: application à la robotique collective. Revue d'intelligence artificielle, 12 (1) (1998) 125-147.
19. Burckert, H.-J. and Fischer, K. and Vierke, G.Transportation Scheduling with Holonic MAS The TeleTruck Approach, Proceedings of the Third International Conference on Practical Applications of Intelligent Agents and Multiagents (PAAM'98), (1998).
20. Ferber, J. Les systèmes multi-agents, Vers une intelligence collective. InterEditions, Paris, (1995).
21. Jonnequin, L., Adam, E., Kolski, C., Mandiau, R. Co-operative Agents for a Co-operative Technological Watch, CADUI'02 - 4th International Conference on Computer-Aided Design of User Interfaces, University of Valenciennes, (2002).
22. Mathieu, P., Routier, J-C., Secq, Y. RIO: Roles, Interactions and Organizations. CEEMAS 2003 : Multi-Agent Systems and Applications III, Lecture Notes in Artificial Intelligence 2691. (2003)

I-MINDS: An Agent-Oriented Information System for Applications in Education

Leen-Kiat Soh, XuLi Liu, XueSong Zhang, Jameela Al-Jaroodi, Hong Jiang, and Phanivas Vemuri

Department of Computer Science and Engineering
University of Nebraska, Lincoln, NE 68588-0115 USA
{lksoh,xuliu,xuzhang,jaljaroodi,jiang,pvemuri}@cse.unl.edu

Abstract. In this paper, we describe an Intelligent Multiagent Infrastructure for Distributed Systems in Education (or I-MINDS) framework that applies an agent-oriented information system for education purposes. Our design employs distributed computing principles (i.e., Java Object Passing Interface and the Distributed Shared Object model) to facilitate cooperation among agents, data gathering and information dissemination, making the infrastructure portable, expandable and secure. Our motivations are to help teachers teach better and students learn better. A teacher, with the help of a teacher agent, is able to address important questions and survey the profile of each student (that is automatically maintained by the teacher agent). A student, with the help of a student agent, is able to collaborate with other students without having to form a buddies group him or herself. Thus, with the agents and a collaborative multiagent system, the I-MINDS framework is able to manage, process and share information intelligently.

1 Introduction

Information technology is rapidly changing the educational process by enhancing the way information and knowledge are represented and delivered to students. The advent of Internet and multimedia technology has meant a potentially drastic change of the teaching and learning process from the traditional classroom setting to a more geographically distributed, virtual but still interactive one. In this paper, we describe the development of an agent-oriented information infrastructure that supports different high-performance distributed applications on heterogeneous systems for a computer-aided, collaborative learning and teaching environment. The infrastructure, called the Intelligent Multiagent Infrastructure for Distributed Systems in Education (or I-MINDS), seamlessly combines an intelligent multiagent model at the application level with a software agent-based distributed computing model at the hardware level to provide functionalities essential in the educational process, such as real-time, as well as offline data and information gathering, analysis and dissemination, embedded feedback, assessment and collaboration. At the low level, we use distributed computing paradigms to build the infrastructure that supports fast, asynchronous and concurrent information processing. At the high level, we employ methodologies in intelligent

P. Giorgini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 16–31, 2004.

agents and multiagent systems to develop software decision makers and monitors. This framework is thus a unified approach to support teaching and learning, from the low-level enabling technology to the high-level cognitive activities.

The long-term goals of our work are two-fold: (1) developing an agent-based distributed computing infrastructure specifically for education, addressing the interaction issues in real-time classroom scenarios, distance learning, and so on, and (2) developing an intelligent multiagent information system, built on top of the infrastructure developed in (1), that is able to monitor the activities, recognize patterns and interact with students and instructors alike to improve the quality of teaching and learning. Thus, our objectives include real-time data gathering, information dissemination and decision making in a classroom setting, utilizing advances in both hardware and software.

Intelligent agents are autonomous and can operate robustly in rapidly changing, unpredictable or open environments. With these intelligent agents serving and catering to students' unique needs and behaviours, the students will be able to participate in a virtual classroom actively rather than listening to the lectures passively as in a traditional virtual classroom. Most agent-based education systems use software agents without fully utilizing the power (or intelligence) of agent-oriented information systems such as the reactivity, pro-activeness and social ability [6]. Moreover, most agent-based education systems are simply a group of non-collaborative, individual agents. Our goal is to also exploit "multiagent system intelligence" to help the transfer of information towards helping teachers teach better and students learn better.

Our Secure Distributed Information (SDI) Laboratory at the University of Nebraska has focused on research in the areas of distributed computation. For I-MINDS, we have incorporated two research products of SDI. First, we employ the Java Object Passing Interface (JOPI) [12] to transfer messages among agents. This interface allows objects to be encapsulated and transferred efficiently. Second, we adhere to a Distributed Shared Object (DSO) model. This model allows us to maintain the coherence and consistency of shared objects in a distributed and collaborative environment. Supported by these two technologies, we are able to enhance the application of our agent-oriented system to collecting, managing, sharing and analyzing data and information in a real-time, dynamic environment. In addition, coupled with multimedia technology, we are able to design an infrastructure that deals with video, audio, images and text as a variety of information sources.

Therefore, our proposed framework is a unique and innovative approach to computer-aided education in that we intend to build the enabling infrastructure so that human factors can be incorporated conveniently into the framework and thus lessen or eliminate some of the weaknesses of the current tutoring systems and multiagent systems for education. Furthermore, the low-level agent-based distributed processing system [13] employs JOPI and DSO to facilitate cooperation among agents, data gathering and information dissemination, making the infrastructure portable, expandable and secure. The JOPI and DSO provide an efficient and seamless interface between the high-level cognitive activities and

the low-level enabling technology. In addition, utilizing Java provides a machine independent system that can be used seamlessly across multiple heterogeneous platforms. Thus, our motivations are to make learning more immediate and responsive, to make teaching more immediate and adaptive and to make knowledge transfer among teachers, among students, and among students and teachers more effective, distributed, and collaborative in a real-time classroom environment.

The outline of the rest of this paper is as follows. In section 2, we provide some related work in the areas of computer-aided education systems. Then, we describe the methodology of I-MINDS as an agent-oriented information system. Subsequently, we present the design of I-MINDS and detail its implementation in Section 4. Finally, we conclude.

2 Related Work

There are in general two approaches to applying agent-based technology in education: (1) as individual intelligent agents such as tutors, and (2) as a group of agents in a multiagent system environment.

The objective of the first approach is to act as assisting software, either as a teacher's helper, or students' tutor. For example, intelligent tutoring systems for algebra, geometry, computer languages (such as PACT [16]), physics (such as ANDES [14][18]) and electronics (such as SHERLOCK [17]) have achieved some level of success in classrooms. Some criticisms of the current state of tutoring systems [15] stem from the lack of sufficient intelligence in the tutoring system necessary to monitor and detect a student's pedagogical behaviour. Students may simply keep guessing until they find an action that gets positive feedback and thus learn to do the right thing for the wrong reasons. The tutoring system will never detect such shallow learning [11].

The objective of the second approach is to provide a computing environment where multiple agents can interact to exchange information so that students or instructors may collaborate on how best to transfer knowledge. In a position paper for the 1997 WWW 6 Workshop on "Teaching and Learning with the WWW: Learning from the Past" (at Santa Clara, CA, April 7-11), Schneider and Jermann discussed three main areas related to such an approach in education: (1) Virtual Campuses, (2) Dynamic Worlds for Learning and Teaching, and (3) Advanced Learning Environments over the Internet. Issues in these areas include multi-user worlds, simulation, accessibility of Web-based teaching, data analysis, and so on. Some have also used an agent-based approach to create virtual libraries where the students share different resources [1]. One key component that is missing in today's multiagent systems in education is to enable the system to utilize and analyze the observed behaviour collected from individual agents and subsequently adapted to such behaviour. That is, most multiagent-based education systems do not handle or deal with data or information among the agents.

In addition to the above, some synchronous virtual classrooms have also been developed [e.g., [2], [3], [4], [5]]. IRI-h [2] is designed to function in a heteroge-

neous network environment and offers audio, video and tool sharing services. It can also support class participants with limited multicast capabilities or limited connectivity bandwidth by providing a scalable infrastructure. Commercial products such as Centra Symposium [3] has features such as structured live interaction, asynchronous learning, rich content support, low bandwidth requirement, enterprise-class management and scalability and is easily deployed. Interwise E-Learning Solution [4] offers one-on-one mentoring sessions, delivers live classes, holds collaborative learning sessions and is able to populate a knowledge repository with on-demand learning objects. Mimio Classroom [5] allows the students to share notes with the instructor in real time and students are able to add their own individual comments and notes, which can be saved and reviewed offline. Our I-MINDS design possesses several of the above features. However, our framework uses agent-oriented mechanisms to better exploit these features to utilize information.

3 Methodology

Figure 1 shows a logical overview of the I-MINDS framework. At the hardware level, we have the participating machines such as a cluster of servers to provide the computing power for the multiagent system and the computing devices used by instructors and students (e.g., laptops, desktops, Smartboards, PDAs). The Agent-based layer provides a seamless interface to all participating machines. JOPI [12] and the distributed shared object model are the interface (i.e., API) for developing applications visible to the user. These applications will utilize the interface to provide the user with a virtual and unified environment for the educational system. Software agents will reside in all participating machines of potentially different platforms, connected via a network, providing the necessary functionality to support the multiagent applications. JOPI allows object-oriented treatment of messages and agent behaviour, adding flexibility and scalability to the system.

3.1 Distributed Computing Infrastructure

The agent-based infrastructure is the backbone of our proposed framework. This infrastructure is a pure Java system based on a distributed memory model. It has a number of components that collectively provide services for high performance educational applications on clusters and heterogeneous systems. At this level, the infrastructure will provide the applications with a seamless access to the different resources available in the distributed environment. This infrastructure will provide automatic application deployment and will support information exchange in the form of objects. This will allow multiple users (students and instructors) to easily access and exchange information. Moreover, JOPI and the distributed shared object model that will be available on the infrastructure will simplify communication and information exchange. To maintain the portability and pure Java implementation of the agents, it has been decided to use the mechanisms provided by standard Java for communication among system components.

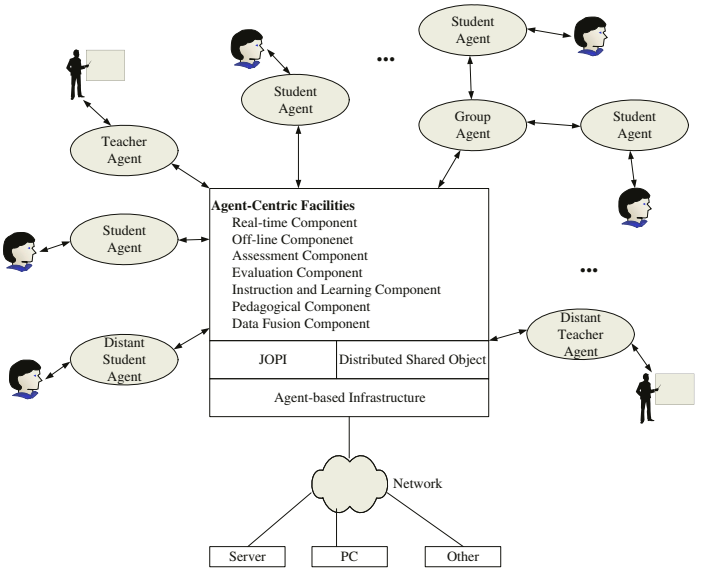

Fig. 1. A logical overview of the I-MINDS framework.

We have identified a set of components for the distributed computing infrastructure: real-time, off-line, distributed, assessment, evaluation, instruction and learning, pedagogical and data fusion components. Some of these components only require one agent to perform the associated tasks, while others require multiple agents to work together to provide application support. Briefly, the real-time component deals with immediate, responsive requirements of the agents. The off-line component allows a teacher agent to review the quality of a previous lecture and the trends of a series of previous lectures. The assessment component assesses students' performance while the evaluation component evaluates teachers' performance. The instruction and learning components lend support via domain-specific advice, suggestions and individualized, performance-based lecture supplements. The pedagogical component looks into the behaviour of the agents and associates patterns to known pedagogical behaviours. In essence, these components form a toolkit that agents can access to obtain information.

3.2 Multiagent System

In our multiagent system, there are a variety of agents: student agents and teacher agents, as well as remote counterparts and group agents. A student agent solicits input from the student, monitors the student's activity, updates teacher's instructions at the student's notebook and reports the student's information to the teacher on demand. A teacher agent collects input from the student agents, monitors the teacher's activity and corresponding students' behaviour, performs a wide range of data analyses on students' input such as sort-

ing, filtering and pattern recognition, and provides decision making support to the teacher. The decision-making activities include data collection and analysis and information dissemination, and strategic planning that adapts to observed classroom behaviour patterns. These agents are distributed among the different participants (students and instructors) within the same location or scattered over a metropolitan or wide area network. A remote agent needs to take into account network bandwidth, information volume and communication delay to ensure seamless participation in the classroom system. In addition, since remote agents may not be in the classroom, additional features such as video and audio streaming are useful. In general, remote agents have tighter time constraints and require a richer set of features. Finally, a group agent is a special student agent that allows a group of students to register their individual student agents. A group agent has a set of group-specific heuristics to guide its activity monitoring, data collection and analysis, as well as information dissemination behaviour within a group.

Briefly, each agent has a set of functionalities such as communication, data processing, a user interface, user registration, real-time monitoring and housekeeping. Communication allows the agent to access remote services; this is achieved using standard Java socket programming. The data processing function includes text parsing, object modeling and even audio and video parsing. The user interface allows a student or a teacher to access the agent software. The user registration mechanism is required to register student agents to the teacher agent and to help with monitoring and housekeeping tasks. For high throughput, each agent is designed to be multi-threaded and thus is able to carry out multiple concurrent tasks. This increases the autonomy and responsiveness of the agents.

3.3 JOPI and Distributed Shared Object (DSO)

We choose JOPI as the underlying language platform as it provides portability, expandability, security, flexibility and resource management. JOPI provides a default security manager used to provide some level of protection for system resources. When the security manager is set for an application (of a infrastructure component), many operations will be checked against the defined security policy before they are executed. For our proposed prototype, different security modes will be mapped to the policies required for each particular application. In this case, it will be possible to grant different access and control levels to different participating groups such as instructors and students. An important reason for adding the security measures is to provide safe utilization of hardware and software resources and to provide protection and privacy to the participants in the system.

Together with our DSO model (currently being built), JOPI provides a consistent platform in which data coherence, address integrity and other system-related parameters are maintained specifically for multiagent environments. This capability allows us to concentrate almost entirely on the development of the multiagent system without having to worry about the low-level computing in-

frastructure that supports the community of agents. Moreover, the distributed shared object model and JOPI allow information and procedures to be passed as objects between agents. This brings forth benefits in two areas: (1) during the design and implementation of the system, and (2) for the extension and monitoring of the system. In other words, JOPI and the distributed shared object model are the enabling technologies for rapid prototyping and development of multiagent systems.

3.4 Agent-Based Computing

In addition to the high-level conveniences (such as autonomy, learning, and reasoning capabilities), the utilization of agent-based computing further enhances the distributed computing environment. The use of agents provides the needed functionality to make the system fully portable due to their communicative ability and degree of autonomy. Agents also provide the system with the flexibility to expand the hardware easily. By activating an agent on a machine, it becomes part of the system where user jobs can execute. This process will be transparent to the user and will not require any changes in the application that uses the system. The user will be able to access and utilize any resources that the agents have access to regardless of their location, hardware architecture or operating environment. This leads to the ability to easily increase the number of participants in the system. Since each agent monitors its own states, each can predicate instances to the applications in the infrastructure, making both the design process of the system and the eventual usage of the applications flexible.

4 Design

4.1 Logical Infrastructure

Figure 2 represents the logical infrastructure of I-MINDS. The network underlying the fundamental communication environment serves as the first level. System-level protocols and encapsulations equip the second level to provide convenient communication and deployment functions to upper levels. To facilitate the communication programming, JOPI and DSO are provided as the third level. Finally, the agents are located at the last two levels. Each agent has two interacting modules: content-independent and content-dependent. The content-independent module provides the definitions and processes for education-related, general services, while the content-dependent module handles specific course-related knowledge base, heuristics and data. This design allows our system to be highly flexible and user-friendly. Also, since I-MINDS is developed using Java, it has high portability and is able to work on heterogeneous environments.

4.2 Topological Structure

Considering that some remote students (and thus their respective student agents) may access the Internet by dial-up with slow access speed, and multi-cast may be prohibited in some network configurations, we design the topological structure of I-MINDS as shown in Figure 3. On the top is the manager of the system,

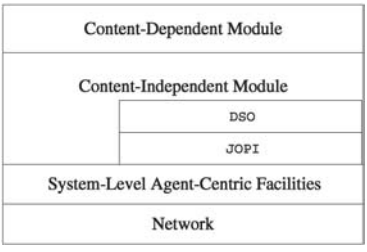

Fig. 2. A logical infrastructure of an I-MINDS agent.

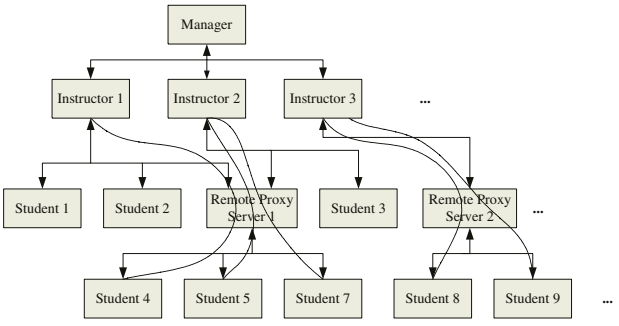

Fig. 3. Topological infrastructure of the I-MINDS multiagent system, an example. Each instructor or student is associated with one agent.

which has the whole information of the system, such as all the courses provided, currently ongoing classes and other static and dynamic information. The instructors are located at the second level and they give the lectures. Besides the students, there are also some remote proxy servers at the third level, which serve as bridges between the teachers and the students where there is low-speed access or where multicast is prohibited in the network. The function of our remote proxy server is similar to Gateways [2]. Students accessing the virtual classroom through the nearest remote proxy servers indirectly (e.g., student 4), as denoted by the dotted, curvy lines.

4.3 Implementation

In this section, we will present the implementation of the manager, the teacher agent, the student agent and the remote proxy server separately, which includes the hardware environment of I-MINDS, the functionalities of the agents and some of the functions provided by I-MINDS. To improve the stability of I-MINDS, we use mature fundamental technology whenever possible. For example, we use the JMF package [8] provided by Sun Microsystems, Inc. to implement the broadcast of video and audio, and JOPI and DSO to handle the communication between the agents.

Manager. This module is one of the agent-centric facilities that manages (1) the course registration of the students through the student agents, (2) the login/logout of the teachers, (3) the IP addresses and listening ports of the teacher agents and (4) the IP addresses of the proxy servers in the system.

Teacher's Site

Teaching Environment

At our teacher's site, we have the following devices:

- In our implementation of I-MINDS, we use Mimio and Mimio mouse [10] to create a convenient teaching environment for the teachers. Mimio is an input device that consists of a capture bar and a stylus, by which everything scratched by the teacher on a whiteboard is captured and digitized automatically and fed into the computer connected to the Mimio mouse.
- We also use a projector to overlay lecture slides on the whiteboard. A teacher is able to write on the whiteboard using the Mimio stylus. Our software superimposed both the lecture slides and the writing by synchronizing the physical coordinates on the whiteboard with those in the digital form stored on the computer. In fact, handwritten text on the whiteboard can be overlaid on anything that is projected to the whiteboard through the computer, allowing I-MINDS to handle both offline and real-time data.
- We also outfit our teacher's site with a microphone and a Webcam to capture the audio and video images of the teacher during a session. The teacher agent transmits the two pieces of data to the student agents. To lower the bandwidth requirement, the teacher agent does not stream the data continuously. Instead, the teacher agent captures the movement of the stylus and sends out the changes between frames.

Teacher Agents

Figure 4 shows the structure of a teacher agent. In the content-dependent module, there are quizzes/exercises and answers from all the students, questions asked by students, rules used for inference and dynamic profiles of the students. The initial database of rules, quizzes and exercises are provided by the teachers or educators. These teaching materials will be modified by the learning mechanism. For example, the rules used to evaluate the quality of the questions can be changed based on its utility. The evaluation mechanism assesses the students based on their responses to the exercises and the quizzes, as well as the monitored questions and actions from their student agents. Based on the profile, the teacher agent will be able to cater to each student agent with a customized set of exercises and quizzes. Meanwhile, the teacher agent maintains a profile of each student. These profiles are also factored into the self-learning activity. Finally, a repository mechanism is included to cache sizable teaching materials into large storage devices for efficient transmission. We aim to support the caching strategy based on the profiled models of the student agents.

Currently, we have fully implemented the teaching environment (See Figure 5 for a snapshot). In addition, we have implemented both the interface modules

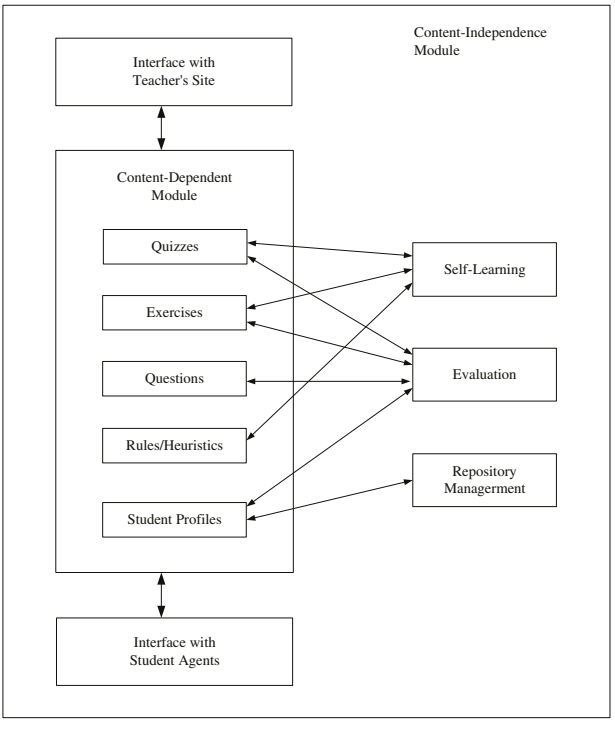

Fig. 4. Structure of our teacher agent.

and part of the evaluation module. We have also implemented a prototype of the student profiling module and a definition of a student profile. Each response by each student through his/her respective student agent is evaluated. The teacher agent presents a ranked list of questions (based on the content of the questions) from the students for the teacher to choose to address. Questions that are picked by the teacher to address will lead to higher "quality" for the students who asked the questions. In this way, the teacher agent learns how to complement the quality of a question with the quality of the student who asked the question. In addition to content-based criteria, the teacher agent also uses heuristics such as "if the student has never asked a question before until now, then rank the current question high." On the other hand, the teacher agent also profiles each student agent: the frequency of the responses received from the agent, the average length, type and quality of the responses, and so on. This profile gives an overall "quality indicator" of each student, which is also utilized in the formation of the "buddies" groups, as described later in the next subsection.

The above design utilizes some principles of agent-oriented information systems. Each agent (student or teacher) is an information system, collecting, transferring and processing data and information. The teacher agent ranks the student agents based on the information it receives from them. Since in a realtime situation, the teacher is not able to answer all questions, for example, asked during

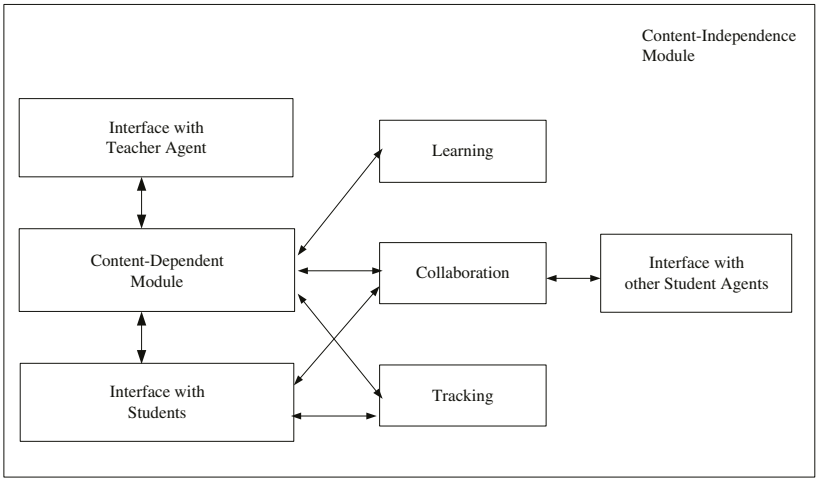

Fig. 5. A snapshot of the teaching environment.

the lecture, the teacher agent ranks these questions based on its profile of the student agents and its content-dependent heuristics, essentially performing dynamic, intelligent information filtering. In this manner, the teacher agent is able to recommend questions that are more important to the teacher to answer in class. Meanwhile, whenever the teacher selects a question, he or she actually teaches the teacher agent that the question is indeed important. This learning by instruction is translated into a modification of the quality indicator of the student agent that submitted the question, which leads to information refinement.

Student's Site

Learning Environment

At our student's site, we have the following application features:

- The student's site application has reception of information in multiple mediums: (1) video which shows the actions of the teacher realtime, (2) content (as images) on the whiteboard (both projection and handwritten text) and (3) audio which carries the teacher's voice.
- The application can also transmit voice to the teacher agent.
- The application supports a multiagent forum. A student can choose to join different fora and communicate with other students during the lecture.
- The application also manages and stores notes for a student. A student can make notes directly on to the lecture images and save them for later use.

Student Agents

Figure 6 depicts the structure of our student agent. After receiving messages and information streams from a teacher agent, the student agent will display

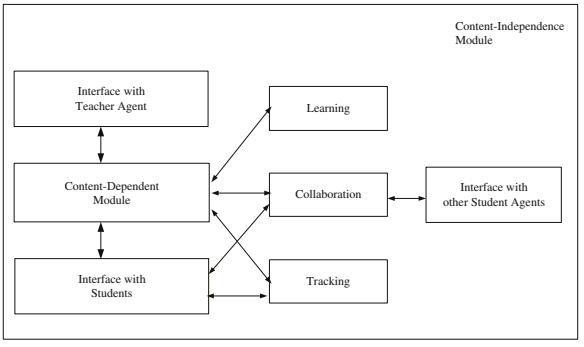

Fig. 6. Structure of our student agent.

them directly to the student. Similarly the student agent will forward the responses from the student to the teacher agent directly.

The tracking mechanism tracks the activities and the study progress of the student. For example, if during a class the student does not touch the keyboard or move the mouse for 5 minutes, then the student agent may play a sound to alert the student to concentrate on class. If the student misses one class, the tracking mechanism may go to the corresponding teacher agent and find out the archived materials for that class according to the syllabus the teacher provides and then remind the student about the missed lectures.

Each agent has a collaboration mechanism[20] that can be activated by a student. When a student asks a question, the student agent sends it to the teacher agent. In addition, the student agent chooses the other student agents (known as "buddies") based on the student agent's profiling of these buddies to send the question to. Thus, buddies may answer questions that the teacher does not respond to in class. Notice that the student agent forms this buddies group automatically and dynamically. Buddies that have not been responsive will be dropped from the buddies group; buddies that have been helpful will be approached more frequently.

Currently, we have implemented the three interfaces, the collaboration module, as well as parts of the content-dependent module and the learning module. Figure 7 shows a snapshot of the learning environment. Our student agents are able to form buddies groups dynamically based on the information shared among the student agents. Note that the information sharing activities among the student agents are behind the scene and the student users do not have access to the shared information. Similar to the agent-oriented design at the teacher's site, these student agents are individual information systems that monitor their respective students, other student agents, and interact with the teacher agent. Each processes the actions of its student and receives teaching materials from the teacher agent. Each profiles other student agents to learn about each other's quality indicators. A poorly-ranked student agent, A, is matched up with a highly-ranked student agent, B, for example, according to the heuristics used in

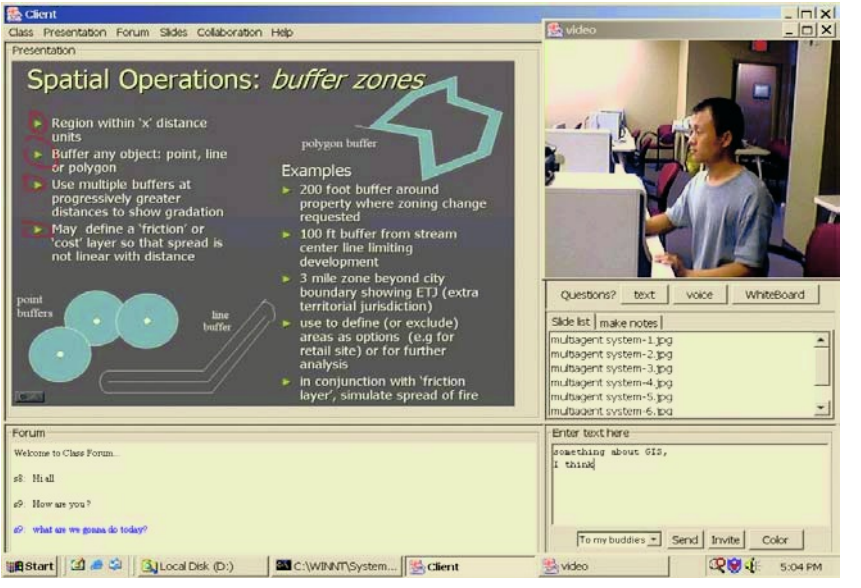

Fig. 7. A snapshot of the learning environment.

the collaboration mechanism. However, if B presents the questions to its student and the student chooses to ignore them, and as a result A receives no responses from B, then A automatically drops B from its buddies group. Thus, our design employs agent intelligence – from its observation of its realtime environment and its built-in heuristics – to form and refine collaborative information teams.

Remote Proxy Servers. The remote proxy servers are responsible for forwarding the data and information from the teacher agents to targetted student agents that have limited Internet connection speed or do not have a multicast capability. Whenever the proxy server receives a connection request from an agent, it spawns a dedicated thread to serve that connection. The proxy server buffers and also processes the data or information before relaying it to the destination agents. For example, it may remove frames from the video transmission to adapt better to the user connection speed.

5 Conclusions and Future Work

The I-MINDS framework has many applications in education, due to its real-time capabilities and agent-based approach, such as real-time in-class instructions with instant data gathering and information dissemination, embedded feedback, unified agent and distributed computing, virtual campus, distance learning, group learning, real-time student response monitoring, performance evaluation and assessment. Our proposed infrastructure can be used to facilitate and enhance the educational process in a distributed environment. For example, it is

possible to support a virtual classroom where students can attend a class and interact in real-time with the instructors and other students from a PC in his/her home. We plan to perform two types of impact analyses: (1) system level and (2) agent level.

At the system level, we want to test in real-time and overload the server to see how well the system responds to various student response behaviours in a classroom setting. From this we will measure the responsiveness and suitability of our system. We also want to investigate the time delay factor and the data volume factor — for example, how exchange of video messages may slow down the system. We also plan to study how distance learning can be synchronized or incorporated seamlessly into in-class teaching and/or learning. We not only want to analyze the participation of the remote students in the class responding to the lecture, but also the participation of the distant students with in-class students as part of a group. We also want to test the scalability and reactiveness of the system. For example, if we have a remote teacher, multiple students thus send information and data to a remote node. How will the distributed computing infrastructure handle this scenario? Basically, at the system level, the objective is to find out how the distributed computing paradigm improves real-time computer-aided teaching and learning in a dynamic classroom setting.

At the agent level, we want to see whether the various agents are able to carry out their tasks. We want to investigate whether a teacher is able to exploit the responsiveness of the teacher agent in data collection and analysis and information dissemination. Similarly, we are interested in the activities of the student agents. We plan to outline the key monitoring statistics such as data volume, response time of a student, the collaborative activities, the number of objects exchanged, how consensus is achieved, how the students make use of the instructions and, most importantly, how information is managed, filtered, exchanged and analyzed. Since each agent keeps a record of its own behaviour, including interactions with the user, we have a wide range of flexibility in our experiments to determine the impact of the intelligent agents and the multiagent system in our system, from non-helpful agents to autonomous ones, from non-communicative agents to peer-to-peer ones. Basically, at the agent level, the objective is to determine how agent-based methodologies facilitate more accurate, effective and efficient data, information and knowledge transfer in a classroom setting.

We have built an I-MINDS prototype and successfully demonstrated the system, in realtime with one teacher agent, delivering a lecture with Power Point slides and online Web pages, and two student agents, with audio, video, image and text exchanged among them. We have built the manager, the teacher's site teaching environment, several modules of the teacher agent, the student's site learning environment and application, and several modules of the student agent. Though not mentioned in this paper, we have also addressed issues in bandwidth and resolution when we implemented our synchronized transmission of video and audio data. Currently, we are building the remote proxy server and embedding executable objects as teaching materials.

To conclude, we have described an I-MINDS framework that applies an agent-oriented information system to education. This system enables teachers to teach better and students to learn better. A teacher, with the help of a teacher agent, is able to address important questions and survey the profile of each student (that is automatically maintained by the teacher agent). A student, with the help of a student agent, is able to collaborate with other students without having to form the buddies group him or herself. The student agent is able to find compatible buddies based on its profiling of other student agents and is able to refine the buddies group based on its realtime observation of the student agents in the group. With the agents and a collaborative multiagent system, I-MINDS is able to manage, process and share information more intelligently, compared to conventional information systems.

Acknowledgment

This work is partially supported by a seed grant from the National Center for Information Technology in Education (NCITE) and a National Science Foundation grant (EPS-0091900).

References

1. G. Kimovski, V. Trajkovic, and D. Davcev: Virtual Laboratory-Agent-based Resource Sharing System, 39th International Conference and Exhibition on TOOLS, 2001, 89-98
2. A. Abdel-Hamid, S. Ghanem, K. Maly, and H. Abdel-Wahab: The Software Architecture of an Interactive Remote Instruction System for Heterogeneous Network Environments, Proceedings. Sixth IEEE Symposium on Computers and Communications, (2001)694-699
3. Centra Symposium 6.0: `http://www.centra.com/products/symposium/info .asp`
4. Interwise E-Learning Solution: `http://www.interwise.com/solutions/elearning.asp`
5. Mimio Classroom: `http://www.mimio.com/meet/classroom`
6. M. Wooldridge and N. R. Jennings: Intelligent agents: Theory and Practice, The Knowledge Engineering Review, (1995) 10(2):115-152
7. I. Greif, (Ed): Computer Supported Cooperative Work: A Book of Readings, Morgan Kaufmann Publishers, 1988
8. JMF homepage: `http://java.sun.com/products/java-media/jmf/`
9. N. Mohamed, J. Al-Jaroodi, H. Jiang, and D. Swanson: JOPI: A Java Object-Passing Interface, Proc. Joint ACM Java Grande-ISCOPE Conference (JGI2002), Seattle, Washington, November, (2002)37-45
10. Mimio Mouse: `http://www.mimio.com/software/index.shtml`
11. Aleven, V., K. R. Koedinger, and K. Cross: Tutoring Answer Explanation Fosters Learning with Understanding, in S. P. Lajoie and M. Vivet (eds.) Artificial Intelligence in Education, (1999)199-206, Amsterdam: IOS
12. Al-Jaroodi, J., N. Mohamed, H. Jiang and D. Swanson: A Comparative Study of Parallel and Distributed Java Projects for Heterogeneous Systems, in Workshop on Java for Parallel and Distributed Computing at IPDPS 2002, Ft Lauderdale, Florida, 2002

13. Al-Jaroodi, J., N. Mohamed, H. Jiang, and D. Swanson: Agent-Based Parallel Computing in Java - Proof of Concept, Technical Report TR-UNL-CSE-2001-1004, 2001
14. Gertner, A. S. and K. VanLehn: ANDES: A Coached Problem-Solving Environment for Physics, in Intelligent Tutoring Systems: 5th Int. Conf., ITS 2000, eds. G. Gautheier, C. Frasson, and K. VanLehn, (2000)133-142
15. Graesser, A. C., K. VanLehn, C. P. Ros, P. W. Jordan, and D. Harter: Intelligent Tutoring Systems with Conversational Dialogue, AI Magazine, (2001)22(4):39-51
16. Koedinger, K. R., J. R. Anderson, W. H. Hadley, and M. A. Mark: Intelligent Tutoring Goes to School in the Big City, J. Artificial Intelligence in Education, (1997)8(1):30-43
17. Lesgold, A., S. Lajoie, M. Bunzo, and G. Eggan: SHERLOCK: A Coached Practice Environment for an Electronics Troubleshooting Job, in J. H. Larkin and R. W. Chabay (eds.) Computer Assisted Instruction and Intelligent Tutoring Systems, (1992)201-238, Hillsdale, NJ: Lawrence Erlbaum.
18. VanLehn, K.: Conceptual and Metalearning during Coached Problem Solving, in Proc. of the 3rd Intelligent Tutoring Systems Conf, (1996)29-47, Berlin: Springer-Verlag
19. VanLehn, K., R. Freedman, P. Jordan, C. Murray, R. Osan, M. Ringenberg, C. P. Ros, K. Schulze, R. Shelby, D. Treacy, A. Weinstein, and M. Wintersgill (2000). Fading and Deepening: The Next Steps for ANDES and Other Model-Tracing Tutors, in Intelligent Tutoring Systems: 5th Int. Conf., ITS 2000, 474-483
20. Ye. Y., E. Churchill, Agent Supported Cooperative Work: an introduction, in Agent Supported Cooeprative Work, Ye and Churchill eds., Kluwer Academic Publishers, (2003)1-25.

Trustworthy Service Caching: Cooperative Search in P2P Information Systems

Yathiraj B. Udupi[1], Pınar Yolum[2], and Munindar P. Singh[1]

[1] Department of Computer Science
North Carolina State University, Raleigh NC 27695-7535, USA
{ybudupi,singh}@ncsu.edu
[2] Department of Artificial Intelligence
Vrije Universiteit Amsterdam, De Boelelaan 1081a
1081 HV Amsterdam, The Netherlands
pyolum@few.vu.nl

Abstract. We are developing an approach for P2P information systems, where the peers are modelled as autonomous agents. Agents provide services or give referrals to one another to help find trustworthy services. We consider the important case of information services that can be cached. Agents request information services through high-level queries, not by describing specific objects as in caching in traditional distributed systems. Moreover, the agents autonomously decide whom to contact for a service, whom to provide a service or referral, whether to follow a referral and whether to cache a service. Thus the information system itself evolves as agents learn about each other and the contents of the caches of the agents change. We study here the effect of caching on service location and on the information system itself. Our main results are that, (1) even with a small cache, agents can locate services more easily; (2) since the agents that cache services can act like service providers, a small number of initial service providers are enough to serve the information needs of the consumers; and (3) agents benefit from being neighbours with others who have similar interests.

1 Introduction

The purpose of information access is to fulfil a need. In conventional practical systems, information access has been mapped to database lookup and an emphasis is placed on looking for correct or relevant results. In contrast, in open settings, we look for authoritative sources that can provide correct and relevant results. The results may be available from multiple sources and hence are not necessarily unique. The above distinction corresponds to the difference between requesting a copy of Abraham Lincoln's Gettysburg Address (of which, say, there is one definitive version) and a commentary on the American Civil War. Or, more prosaically, a copy of the first draft WSDL standard versus an analysis of Web Services. Importantly, in an open environment, we cannot rely on traditional mechanisms using regulatory restrictions for ensuring the quality of the services obtained or the trustworthiness of the peers found over the network.

Thus, the need is to develop a decentralized approach wherein peers can find peers who are trustworthy and offer high quality services. Some of the peers would be service providers, possibly specializing in apparently arcane services and catering to a

P. Giorgini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 32–44, 2004.

niche clientele. These peers could control and exploit private knowledge bases that are unreachable by traditional search engines. Other peers would learn about and use the above peers. If a peer finds the received services useful, it may (1) use it (2) use it and refer others to it, or (3) use it and cache it so as to serve others from its cache. Caching aids the search for information since a peer that is looking for information can find it in a nearby cache rather than having the original provider generate it again from scratch. This way, by mutually helping each other to find trustworthy services, the peers can induce the formation of communities of common interest and practice.

A number of approaches have been developed for caching information in traditional distributed systems. Even those developed for P2P systems tend to take a traditional stance in that they view information access primarily as looking up specific objects rather than fulfilling information needs. However, a suitable flexible caching technique must satisfy two important criteria:

- *Peer autonomy*. A peer should have the freedom to choose with whom it interacts and how it carries out its interactions. It should also be able to change its neighbours to get the best results.
- *Peer heterogeneity*. Peers can be heterogeneous by having different policies and offering services distinct from all others. By accommodating heterogeneity, an information system also accommodates the fact that the peers can offer different levels of trustworthiness and be perceived by others as having different levels of trustworthiness.

Organization. Section 2 introduces key elements of our referrals-based model for service caching. Section 3 describes the evaluation scheme, discusses some of the important control variables used in evaluation and presents our results. Section 4 concludes with a discussion of some relevant literature and of directions for further work.

2 Marmara: Flexible Caching Technique

We provide some essential background on Marmara, a flexible caching technique based on *m*ulti-*a*gent *r*eferrals [1]. Each agent offers varying levels of trustworthiness to others and is potentially interested in knowing if other agents are trustworthy (for its purposes). The agents can keep track of each other's trustworthiness by flexibly giving and taking *referrals* and by learning about each other and the available services. There is no guarantee about the quality of a service provided by any agent or the suitability of a referral it offers. Hence, we do not assume that any agent should necessarily be trusted by others.

2.1 Referrals-Based Protocol

When an agent anticipates the need for a service, the agent begins to look for a trustworthy provider for the specified service. The agent queries some other agents from among its *neighbours*, which are a small subset of the agent's acquaintances. A queried agent may offer its principal to perform the specified service or may give referrals to agents of other principals. The querying agent may accept a service offer, if any, and may pursue referrals, if any. Each agent maintains models of its acquaintances, which describe their *expertise* (i.e., the quality of the services they provide), and *sociability* (i.e., the quality of

the referrals they provide). Both of these elements are adapted based on service ratings from the agent's principal. The interests and expertise of the agents are represented as term vectors from the vector space model (VSM) [2], each term corresponding to a different domain, whereas the sociability is denoted with a scalar.

Algorithm 1 Ask-Query().

```
 1: if (hasCachedAnswer) then
 2:     Evaluate cached answer
 3:     Determine if it has to stay in cache and then return
 4: else
 5:     while (more matching neighbours to ask) do
 6:         Send query to matching agents
 7:         Receive message
 8:         if (message.type == referral) then
 9:             Send query to referred agent
10:         else
11:             Add answer to answerset
12:         end if
13:     end while
14:     for i = 1 to |answerset| do
15:         Evaluate answer(i)
16:         Cache answers
17:         Update agent models
18:     end for
19: end if
```

Each agent is initialized with the same model for each neighbour, this model being rigged to encourage the agents to both query and generate referrals to their neighbours. An agent that is generating a query follows Algorithm 1. Since we do not have actual principals (i.e., humans) in the evaluation, the queries and the answers are generated by the system. More precisely, an agent generates a query by slightly perturbing its interest vector, which denotes that the agent asks a question similar to its interests. Next, the agent checks its own cache if it has a similar query cached already (line 1). If there is no answer found, the agent sends the query to a subset of its neighbours (line 6). The main factor here is to determine which of its neighbours would be likely to answer the query.

One way to choose the neighbours is through the capability metric, which measures how sufficient the expertise vector is for a given query vector [3,4]. Essentially, it resembles cosine similarity but it also takes into account the magnitude of the expertise vector. This means that expertise vectors with greater magnitude indicate higher capability for the given query vector. In Formula 1, Q ($\langle q_1 \dots q_n \rangle$) refers to a query vector, E ($\langle e_1 \dots e_n \rangle$) refers to an expertise vector and n is the number of dimensions these vectors have.

$$Q \otimes E = \frac{\sum_{t=1}^{n} (q_t e_t)}{\sqrt{n \sum_{t=1}^{n} {q_t}^2}} \tag{1}$$

If $Q \otimes E$ is greater than a threshold, for two agents A and B, then B is capable of answering A's query. Keeping the threshold high results in choosing only the most

capable agents. Figure 1 shows an example network, where the nodes denote the agents and a dotted line from an agent A to an agent B means that B is a neighbour of A. The vectors marked with I are the actual interest vectors of the agents.

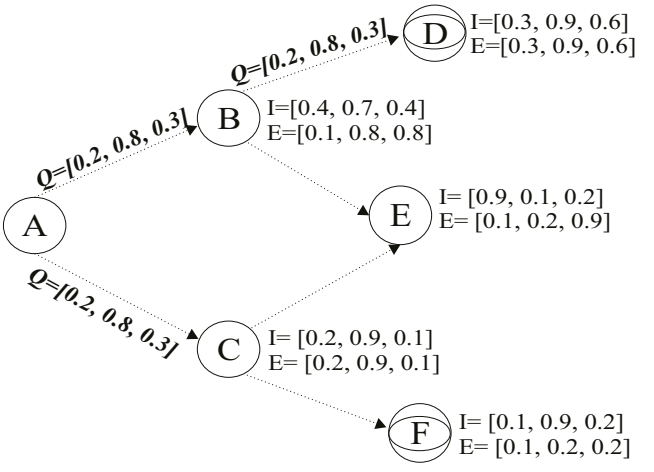

Fig. 1. An example search.

An agent that receives a query acts in accordance with Algorithm 2. An agent answers a question if its expertise matches a question. If the expertise matches the question, then the answer is the perturbed expertise vector of the agent. When an agent does not answer a question, it decides if it is interested in the query (line 5). If it is interested, then it applies the Ask-Query() method (Algorithm 1) to forward the query, find the answer itself, and return the answer it gets. Otherwise, it chooses some of its neighbours to which to refer.

Algorithm 2 Answer-Query().

```
 1: if (hasCachedAnswer) then
 2:     Return cached answer
 3: else if (hasEnoughExpertise) then
 4:     Generate answer
 5: else if (interestedInQuery) then
 6:     Ask-Query()
 7:     Return answer
 8: else
 9:     Refer neighbours
10: end if
```

Back in Algorithm 1, if an agent receives a referral to another agent, it sends its query to the referred agent (line 9). After an agent receives an answer, it evaluates the answer by computing how much the answer matches the query (line 15). In a real application, user feedback would be involved here but, in the simulation, the agent itself evaluates

the answer. Thus, implicitly, the agents with high expertise end up giving the correct answers. After the answers are evaluated, the agent caches only the useful (good) answers (line 16). When a good answer comes in, the modelled expertise of the answering agent and the sociability of the agents that helped locate the answerer (through referrals) are increased. Similarly, when a bad answer comes in, these values are decreased (line 17).

Each agent has equal number of neighbours. At certain intervals during the simulation, each agent can choose new neighbours from among its acquaintances. Usually the number of neighbours is fixed. Therefore, adding new neighbours means dropping some existing neighbours. When choosing neighbours, the acquaintances are ranked based on an equal weight of their modelled expertise and the sociability. The top ranked agents become the neighbours. In other words, it is best to interact with agents that can provide useful answers or useful referrals or both.

As the agents change neighbours, they find neighbours that best suit their interests. When searching a service, each agent follows referrals from its neighbours, who in turn give referrals to their neighbours, and so on. Since each agent follows referrals from the agents that it trusts, it is possible to locate trustworthy providers.

2.2 Cache Properties

In Marmara, a cache consists of a set of entries that contain the data item as well as some information about the quality or the appropriateness of the given data item. The query is modelled as a vector by modelling different domains as different dimensions of the vector. Thus, an agent can specify the different keywords of search, which can be represented by different values for the different dimensions of the vector. The answer is again a vector that represents the different domains of the query. The quality of the cache entry is modelled as to its appropriateness to the owner of the cache based on its interest.

The peers insert items into their caches based on their own interests. The sizes of the caches are usually limited. Hence, a cache entry can be automatically replaced with a new entry based on the cache replacement policy in use. An agent can also delete an entry due to lack of interest in the item. Even after the owner of a cached entry deletes the original item, the peers that have a cached copy of the item are free to keep the item in their caches. An agent formulates queries by specifying a list of keywords which will be translated into a query vector. The keywords-based query gives flexibility to the search, in that the source peer does not need to know the exact name or the unique identifier of the data item it is looking for.

Let's walk through an example. Agent A is looking for an item that can be cast into a query vector of $[0.2, 0.8, 0.3]$. The peers D and F have the desired item in their caches. Following Algorithm 1, A starts the search by computing the query vector. Next, it decides to which of its neighbours (B, C) to send the query, using (Equation 1). Even with a fairly high threshold, both B and C qualify. So, A sends its query to both B and C as indicated by the query vector on the links from A to B and A to C. Agents that receive the query follow Algorithm 2. Accordingly, when B receives the query, it checks its own cache to see if the item exists there. Not finding it in its own cache, B decides to send the query to its own neighbours, since its interest is similar to the query, i.e., it is curious about the results itself. After B obtains the desired item itself, it can forward

it to A. Notice that among B's neighbours (D, E), D is the only one whose expertise vector matches the query, so B sends the query only to D. After B receives the item from D, it puts the item into its cache and forwards it to A. Meanwhile, since C is not interested in the query, it just sends a referral back to A.

Notice that a search can yield multiple results. Some of the peers may have more than one item whose content matches the query vector. For each item, we can specify how good a match the item is to the given query, which allows the returned items to be trivially ranked based on the strength of their apparent match.

3 Evaluation

We evaluate Marmara using simulations over an agent platform, which enables the agents to query each other, obtain responses or referrals and cache useful answers. We now briefly describe the simulation environment and some of the important parameters involved in testing.

Our simulation contains 400 agents and 4 domains in the vector model. There are 20 or 40 *experts* in the system (5 or 8 in each domain) who provide services — these are the agents who originate the information that others may cache. Experts give good answers in their domains of expertise. The remaining agents are service consumers, whose interests may span several domains. Service providers do not generate queries and therefore do not have any neighbours. The service consumers all have four randomly chosen neighbours. During the course of the simulation, after every three queries each agent has a chance of changing neighbours. The simulations are run for the duration of 10 neighbour changes.

3.1 Control Variables

We describe here some of the variable parameters used in our study and discuss how they affect the functioning of Marmara.

- **Threshold for forwarding a query:** When an agent receives a query that it is also interested in, it can search for the answer itself. We compute the necessary level of interest by using the capability metric between the query and the interest vector (Algorithm 2, line 5). If this similarity is greater than the threshold for forwarding a query, the agent performs the search itself. Setting high values for this threshold results in a situation when the agent will search only for queries that are highly similar to its own interests, whereas setting low values results in a situation where the cache would have answers to a broad area of interests. A low value for the threshold reduces its usefulness to the agent's own principal. Therefore, we set the threshold high so that most of the forwarded queries will closely match the agent's own interest.
- **Cache size:** This is the number of entries that can be stored by an agent in its cache. This factor plays a big role in the system. Having unlimited or a high cache size results in every good answer being cached in the system. This results in a majority of answers being directly retrieved from the agent's own cache, leading to a reduced

exploration of the system. Limiting the cache size means that we will encounter cache overflows to be addressed according to a cache replacement policy (discussed next). Our experiments consider caches of size 15, 30 or unlimited.

- **Cache replacement policy:** When items in the cache are replaced, it is important to maintain cache quality, i.e., the quality of the answers to their corresponding queries stored in the cache. In our initial experiments, we experimented with a *random replacement policy*, which randomly replaces any entry in the cache whenever there is an overflow. When this policy is in effect, we observe that the cache quality of the system varies randomly. Hence, we consider a more efficient policy which uses the quality of each entry before deciding the entry to be replaced. This *quality based replacement policy* replaces the entry with the least quality if the quality of the entry to be added is higher than the least one. When this policy is used, we observe that with more replacements the overall quality of the cache improves.
- **Fidelity:** This threshold value is used in looking up the cache for a matching query. If the similarity between the current query and a previous query exceeds this threshold, then we consider that there is a cache hit. If there are multiple hits, the best cached answer is returned. A value of 1.0 for fidelity means an exact match between the queries is required, which would be rare, but a high fidelity improves the quality of the answers received. Hence, we keep the value of fidelity in the range 0.9 to 1.0.

3.2 Results

We evaluate this caching technique by comparing the performance of the system with and without caching and by varying some of the control parameters discussed above. The evaluation depends on some measures of the performance and structure of the information system.

Success Rate. This measures the ratio of the queries for which at least one good answer is found. Let T be the total number of queries. Let G be the total number of queries for which at least one good answer is found. Then success rate is given by G/T. The success rate is calculated for each agent for one neighbour change of the simulation and then averaged over all service consumers. Since the service providers do not generate any queries, they are not factored into the calculation. An agent can find a good answer for its query, either from other agents or from its own cache. Initially, more answers are found through other agents. As agents fill their caches, more answers are retrieved from the caches, without the agents having to perform another search. This enables the agents to reach answers faster than before.

As it is not practical to have an unlimited cache size, we limit the cache size and study the system. Figure 2 plots the success rate for different levels of caching when the population has 20 experts. The line labeled *Without caching* shows the success rate of the system when there is no caching. The remaining lines denote the success rates with different cache sizes. Even with a small cache size (size 15), the number of good answers received increases tremendously compared to the system without caching, as seen by the jump of success rate in Figure 2.

There is a big improvement in the success rate when we change from no caching to a cache size of 15. However, the success rate does not improve as much when the

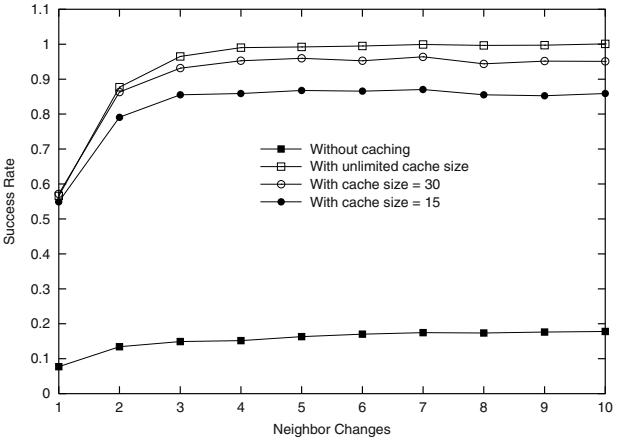

Fig. 2. Effect of caching on success rate.

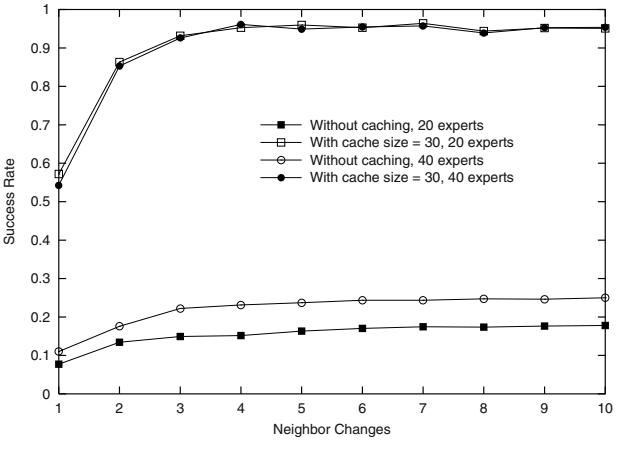

Fig. 3. The effect of the number of experts on success rate.

cache size is increased further. For example, the success rate improves only a little when the cache size is increased to 30 and only slightly more when the cache is allowed to be unlimited. This saturation can be explained by the fact that after a certain size, the number of good answers retrieved from the cache tends to be the same.

Effect of the Number of Experts on Success Rate. Figure 3 plots the success rates for the case without caching and with a caching size of 30, both with 20 experts and 40 experts in a system. In the systems with caching, the lines for the cases of 20 experts and 40 experts overlap. This means that the number of experts in the system does not really affect the success rate of a system with caching. On the other hand, there is an improvement in success rate for the case with 40 experts over 20 experts in a system

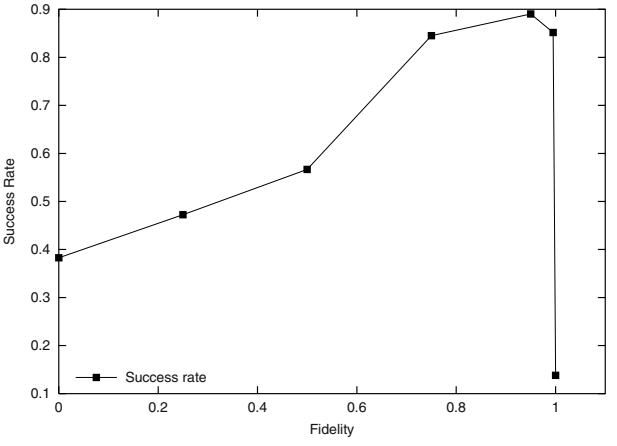

Fig. 4. The effect of fidelity on success rate.

without caching. We obtain similar results with unlimited caching and a cache size of 15. Intuitively, in a system with caching, there is a reduced dependence on finding an expert since after a while many service consumers have good answers in their caches. That is, any agent who returns an answer from its cache takes on the role of a quasi-expert, even though it may not truly be an expert.

Effect of Fidelity on Success Rate. The fidelity threshold plays a crucial role in determining the accuracy of the results returned from the cache. Figure 4 plots the success rate for different values of fidelity in a system with 20 experts and a cache size of 30. We observe that the success rate keeps deteriorating as we lower the value of this threshold. On the other hand, we cannot have a perfect value of 1.0 for this threshold as the queries will have to be matched perfectly to be retrieved from cache and this happens rarely if at all. There is a drastic drop in the success rate for 1.0 as it reduces to the case of no caching.

Coverage and Correctness. The following two measures guide the behavior of success rate when fidelity is varied. Let G and T be as in the definition of success rate. Let A be the total number of queries for which an answer is found. *Coverage* is the ratio of the total number of queries answered to the total number of queries posed, i.e., A/T. *Correctness* is the ratio of the total number of queries with a good answer (i.e., answered correctly) to the total number of queries answered, i.e., G/A. The product of these two measures gives the success rate.

Figure 5 plots coverage and correctness against fidelity. We observe that coverage does not vary much as we increase fidelity, but there is a sudden decrease when fidelity approaches 1.0. Correctness increases steadily with fidelity, but this also drops suddenly for the fidelity value of 1.0. This can be explained with the following. When the fidelity is 1.0, the system is essentially reduced to the case of no caching; hence, a low number of queries are answered as observed earlier in Figure 2. Comparing Figure 4 and Figure 5,

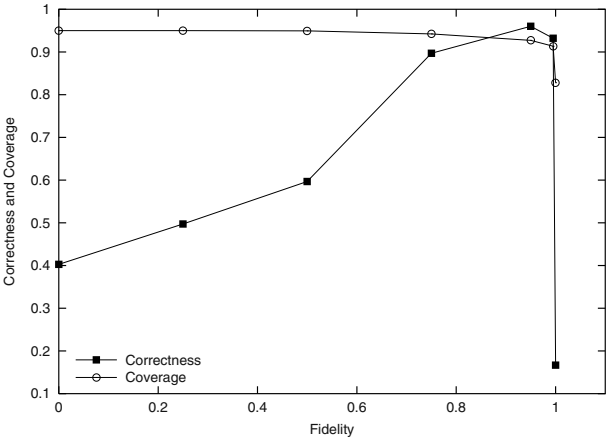

Fig. 5. The effect of fidelity on coverage and correctness.

we observe that the success rate increases gradually with the increase in correctness. It reaches a maximum when the correctness reaches a maximum, while the coverage does not vary much. When fidelity approaches 1.0, as both correctness and coverage see a sudden drop, the success rate also drops suddenly.

Interest Similarity. Interest similarity measures how similar the interests of two agents are. Again, we use a metric that operates on the interest vectors of the agents.

$$I_A \oplus I_B = \frac{e^{-\|I_A - I_B\|^2} - e^{-n}}{1 - e^{-n}} \tag{2}$$

In Equation 2, I_A and I_B denote the interest vectors of two agents A and B, respectively. Here n is the length of the interest vectors. The metric captures the Euclidean distance between the two vectors and normalizes it to return a value between 0 and 1.

For each agent, its similarity to its neighbours is calculated. Then, the average over all the agents is taken. Figure 6 plots the variation of interest similarity for different levels of caching in a system with 20 experts. (Our results for 40 experts are similar.) We observe that interest similarity increases with every neighbour change. This means that the agents tend to form neighbours with those agents who have similar interests. The intuitive explanation of this is that when two agents A and B have similar interests, they tend to generate similar queries. If A has cached an answer to a query that is later also generated by B, B can locate A as a good provider. Hence, A's cache provides an incentive for B to choose A as a neighbour.

We also observe that the values of interest similarity in the cases with caching are higher than the case without caching. We conclude that that caching has resulted in the agents with similar interests to group together. This situation could lead to some interesting observations such as the formation of communities in the network of agents.

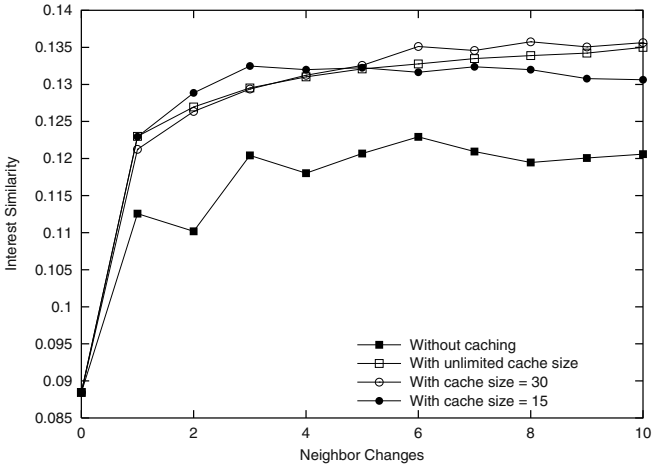

Fig. 6. The effect of caching on interest similarity.

Local Quality. The *local quality* viewed by an agent reflects the usefulness of the neighbours of the agent, given its interest and their expertise. That is, we estimate the likelihood of the neighbours themselves giving good answers to the questions and ignoring the other agents. We calculate local quality as obtained by an agent and then average it over all agents.

Figure 7 plots local quality for different levels of caching when the population has 20 experts. We see that there is a gradual decrease in the local quality of the network. When agents choose neighbours they take into account that good answers were received from an agent and cannot differentiate whether the answer was given from a cache or was generated afresh. Hence, an agent that serves items from its cache may be pointed at by many even though it is not an expert itself but has answers to the queries that the other poses.

4 Conclusion

Our referrals-based caching model that takes an adaptive, agent-based stance on peer-to-peer information systems. Referrals are a natural way for people to go about seeking information. One reason to believe that referral systems would be useful is that referrals capture the manner in which people normally help each other find trustworthy authorities. MINDS, based on the documents used by each user, was an early agent-based referral system [5]. Kautz *et al.* model social networks statically as graphs [6]. They study properties of these graphs such as how the accuracy of a referral to a specified individual relates to the distance of the referrer from that individual. The above approaches do not incorporate caching.

Recently, several peer-to-peer network architectures have been proposed [7,8] and, using these as underlying layers, file-sharing application have been built [9,10]. Rather than allowing peers to autonomously decide what to cache or index, these systems model the network as a distributed hash table that maps keys to peers. Each data item

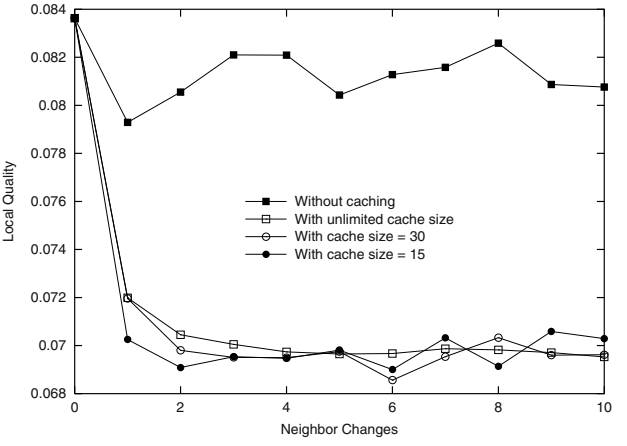

Fig. 7. The effect of caching on local quality.

is replicated an equal number of times and the assignment is not decided by the peer. Furthermore, as long as the peers in the system do not change, the search route is always the same (i.e., the agent has to contact the same agents).

This contrasts with Marmara, which allows agents to cache items that are of interest to them. The items that are of interest to more are cached by more peers. Furthermore, based on the previous interactions with others, each agent chooses the agents with which it wants to interact. The agents that provide better services, either by generating an answer or by serving an answer from their caches, are preferred over others. Hence, each agent adaptively decides on its neighbours.

Aberer and Despotovic develop a reputation-based trust model to manage the trustworthiness of agents in a peer-to-peer system [11]. The agents that have been cheated file a complaint about the other party. The complaints are stored in a P-Grid structure [12], a distributed structure that is maintained by multiple parties. An agent that is looking for a trustworthy agent pulls the complaint entries from the P-Grid and aggregates them. In our approach, the agents that provide evidence are rated. Hence, agents adapt by choosing neighbours that are most useful to them. Contrary to aggregating evidence from all possible agents, the agents that have not been useful in previous interactions are not considered in future interactions.

Marmara allows richer queries to be formulated, allowing trade-offs between search costs and item quality. In future work, we will perform experiments that exploit these trade-offs. We will also study other aspects that affect the performance of the system, such as cache replacement policies and local policies of the agents. A problem of particular interest is about revoking answers or letting cached answers expire. If the essential updates or revocations can propagate through the system, it would produce better responses without compromising the overall quality.

Acknowledgments

This research was supported by the National Science Foundation under grant ITR-0081742. We thank the anonymous reviewers for helpful comments.

References

1. Yolum, P., Singh, M.P.: Flexible caching in peer-to-peer information systems. In: Proceedings of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS). (2002) 72–83
2. Salton, G., McGill, M.J.: An Introduction to Modern Information Retrieval. McGraw-Hill, New York (1983)
3. Yu, B., Singh, M.P.: Searching social networks. In: Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), ACM Press (2003) To appear.
4. Singh, M.P., Yu, B., Venkatraman, M.: Community-based service location. Communications of the ACM **44** (2001) 49–54
5. Bonnell, R., Huhns, M., Stephens, L., Mukhopadhyay, U.: MINDS: Multiple intelligent node document servers. In: Proceedings of the 1st IEEE International Conference on Office Automation. (1984) 125–136
6. Kautz, H., Selman, B., Shah, M.: ReferralWeb: Combining social networks and collaborative filtering. Communications of the ACM **40** (1997) 63–65
7. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for Internet applications. In: Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, ACM (2001) 149–160
8. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. (2001) 161–172
9. Dabrek, F., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I.: Wide-area cooperative storage with CFS. In: Proceedings of the ACM Symposium on Operating System Principles (SOSP). (2001) 202–215
10. Rowstron, A., Druschel, P.: Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In: Proceedings of the ACM Symposium on Operating System Principles (SOSP), Banff, Canada (2001) 188–201
11. Aberer, K., Despotovic, Z.: Managing trust in a peer-2-peer information system. In: Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM). (2001) 310–317
12. Aberer, K.: P-Grid: A self-organizing access structure for P2P information systems. In: Proceedings of Cooperative Information Systems (CoopIS). (2001) 179–194

Agent-Based Support for Mobile Users Using AgentSpeak(L)

Talal Rahwan[1], Tarek Rahwan[2], Iyad Rahwan[3], and Ronald Ashri[1]

[1] School of Electronics and Computer Science
University of Southampton, Southampton, UK
{tr03r,ra}@ecs.soton.ac.uk
[2] Faculty of Computer Engineering
University of Aleppo, Aleppo, Syria
tarek_rahwan@hotmail.com
[3] Department of Information Systems
University of Melbourne, Melbourne, Australia
i.rahwan@pgrad.unimelb.edu.au

Abstract. This paper describes AbIMA, an agent-based intelligent mobile assistant for supporting users prior to and during the execution of their tasks. The agent is based on the well-known AgentSpeak(L) agent architecture and programming language, which provides explicit representations of agents' beliefs, desires and intentions (BDI). AbIMA is implemented using Java 2 Mobile Edition and is tested on a hand-held computer. We also provide conceptual foundations and discuss various challenges relating to the use of cognitive agent architectures for intelligent mobile user support.

1 Introduction

In recent years, the use of mobile handheld computing devices has been on the increase. They offer a wide variety of services to mobile users, such as information provision and management [22]. However, there are still many opportunities for providing more sophisticated, *intelligent* services to users on the move. Such services should go beyond basic information provisioning and organisation tasks by providing intelligent, pro-active support to mobile users prior and during the execution of their tasks.

Consider an engineer following a project agenda, which involves the completion of a variety of tasks. Suppose that the engineer is using a mobile computing device to help manage the work agenda, conduct meetings with clients and consultants, travelling to a variety of building sites, and so on. In the process of task execution, the engineer may face unexpected situations. For example, a meeting may take longer than expected, or a technical problem on site may preclude the completion of one stage in the building process. A useful intelligent support system must be capable of dealing with such unexpected situations and provide the user with alternative *plans* for achieving the objectives. Such a system must be capable of acquiring information about the environmental context, the objectives required, the alternative means through which such objectives may be achieved and, most importantly, it must be able to reason about all of these concepts in order to provide simple, coherent support to the user. Agent-based approaches have

P. Giorgini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 45–60, 2004.

become increasingly popular due to their ability to produce modular software systems capable of providing intelligent assistance in dynamic, unpredictable environments [11].

An intelligent agent is an agent that is capable of flexible autonomous behaviour [11], where flexible means:

1. *Responsive (or reactive)*: able to perceive the environment and respond in a timely fashion.
2. *Proactive*: exhibit goal-directed behaviour and take the initiative when appropriate.
3. *Social*: able to interact with other agents or humans when needed.

Based on this underlying understanding of agent software, agents seem to offer a set of capabilities that are very closely aligned with the requirements of applications for mobile users. This is true for the following reasons. Firstly, a mobile user is usually situated in some environment, which can be represented in terms of context information, such as the time, place and task at hand. Secondly, the environment is dynamic, since users may move from one place to another and since their tasks may change based on their circumstances. Following the earlier example, the engineer may move from one site to another and may have different tasks to achieve etc. Thirdly, a mobile computer system must have the ability to be proactive, reasoning about the user's goals and how they may be achieved. Finally, in mobile settings, there is a need for the ability to interact with the user and, potentially, with other agents. For example, an agent working on behalf of an engineer may interact with agents representing other engineers to sort out meetings and schedule joint tasks.

Some of the most successful theoretical frameworks of rational agents are those belonging to the family of Belief-Desire-Intention (BDI) architectures [5, 17, 18]. It has been argued that these three elements of an agent's mental state can provide a basis for rational action. The agent has explicit representations of its beliefs about itself and its environment, its desires (or goals)[1], which are states of the world it seeks to bring about, and its intentions, which are active plans that the agent adopts in an attempt to achieve these desires. BDI agents have proven useful in the theoretical study of rational agents [21] and in practical applications, for example, in the telecommunications industry [14] and the defence industry [8]. Even though there is a body of research on implementing agents on mobile and handheld devices (e.g., [1, 12]), no attempt, to our knowledge, has been made to implement a specific logic-based BDI architecture on these devices.

This paper represents a comprehensive attempt to implement a Belief-Desire-Intention agent architecture and programming language, specifically the AgentSpeak(L) architecture [17], on a mobile device. By doing this, the project advances the state of the art in two ways: (i) the project investigates how a cognitive model of computational agency and, in particular, how a BDI framework may assist in building applications that provide useful, intelligent support for mobile users in realistic settings. (ii) In doing so, the project also attempts to tackle challenges relating to automated support and volatile user behaviour. This is demonstrated through a pilot agent application for supporting a student in completing a number of tasks on campus.

This paper, which is an extended version of [16], is organised as follows. In the next section, we present a scenario involving a student attempting to complete a number of

[1] In the remainder of the paper, we shall use the terms 'desire' and 'goal' interchangeably.

tasks on campus. We use this scenario to extract some of the core requirements needed in an intelligent mobile software assistant. In section 3, we give a brief introduction to the AgentSpeak(L) agent programming language, showing how it has been used in a novel way to provide intelligent assistance. In section 4, we demonstrate AbIMA through a simple illustrative example, showing the user interface and screen-shots. Section 5 shows some of the emerging challenges that we have encountered and which provide fertile ground for future research. We finally discuss related work in section 6, followed by a conclusion in section 7.

2 Motivations and Design Requirements

In this section, we explain in more detail the reasoning behind the adoption of an agent-based approach to support mobile users. In particular, we present a specific scenario in which it would be helpful to provide task support for the user. Subsequently, we outline the essential features that need to be provided by the support system and explain how a BDI agent could provide some of the required functionality through its interactions with the user.

2.1 Scenario: Student on Campus

We begin by outlining a specific scenario through which we can extract the essential requirements for task support for a mobile user in a dynamic environment. Consider a student, named Omar, on his first day at the university who needs to achieve the following major objectives during the day:

A. Go to the Faculty of Computer Engineering building at the University.
B. Attend a lecture for the subject "Introduction to Computer Science" from 10:00am to 11:30am.
C. Get his new student card from the university registration department.
D. Meet with his friend Ziad on the way to the Grand House Restaurant for lunch. Ziad studies architecture and will finish his lecture at 12:30pm.

Within an agent context, the above tasks may be represented as a set of goals that need to be fulfilled in the given sequence. In order to fulfil each goal, Omar needs to execute a sequence of actions (i.e. to execute a plan). There might be a number of plans for achieving the same task. For example, in order to achieve the first goal, there might be two possible plans:

Plan A.1:
A.1.1: Wake up at 09:00am;
A.1.2: Get a taxi;
A.1.2: Ask the taxi to go to the Faculty of Computer Engineering.

Plan A.2:
A.2.1: Wake up at 8:15am;
A.2.2: Get a service bus from home to the main City Square;
A.2.3: Get another service bus from the City Square to the University Square;
A.2.4: Walk up the university road until you find the building.

The plan Omar will actually follow will depend on a number of factors, such as his preference for what time to wake up, his current budget and so on. As a result, Omar needs to perform *plan selection* based on some preference criteria. Suppose Omar chooses to use plan A.2. This plan now becomes an *intention*. In other words, Omar intends to execute this plan in order to achieve his goal.

Now, in order to achieve the task A.2.1 of waking up at 8:15am, Omar might set up the alarm clock. Then, he needs to get a service bus to the City Square. This might, again, require the execution of another set of sub-tasks. Omar might consider going to the City Square as a *sub-goal* that is part of his intention towards achieving the *super-goal* of going to the Faculty of Engineering building. Finally, in order to achieve this sub-goal, there might be a number of different possible plans to intend:

Plan A.2.2.a:
A.2.2.a.1: Go to Teshreen Street and board service bus number 12.
A.2.2.a.2: Get off at the City Square.

Plan A.2.2.b:
A.2.2.b.1: Go to Nile Street and board service bus number 8.
A.2.2.b.2: Get off at the City Square.

In this manner, Omar can continue, attempting to achieve all of his goals by executing the appropriate plans, which may trigger other sub-goals, and so on.

In the process of executing his plans, however, a number of problems may arise. For example, suppose that after reaching the City Square, Omar is not able to take a bus to the University Square because the bus line is not operational (say because the roads are closed for a diplomatic visitor). In such a case, Omar should be able to find an alternative plan for reaching the university, say, by taking different bus lines. In other words, Omar needs to perform some form of *plain failure recovery*. The new alternative plan must take into account the new context.

Another potential problem Omar might face is *conflict between different goals*. Suppose, for example, that Omar wishes to play football with his friends that morning. Obviously, this goal conflicts with the goal of attending the morning lecture. Omar might be able to *resolve* that conflict by arranging for a different time to play football or might have to perform *goal selection* in order to make a decision about which goal is more important.

2.2 Requirements for Intelligent Mobile Assistance

It is clear from the scenario we presented in the previous subsection that as the number of tasks and alternative plans involved increases, the complexity of the user's agenda increases significantly. This creates the opportunity for providing automated support for the mobile user. In this paper, we are interested in supporting the user by providing appropriate *advice*, in the form of suggested plans of action, throughout the execution of his/her tasks. Such support must overcome particular challenges arising due to the dynamic nature of the environment and the mobility of the user. The following are the core essential features that a software system providing support to mobile users must provide.

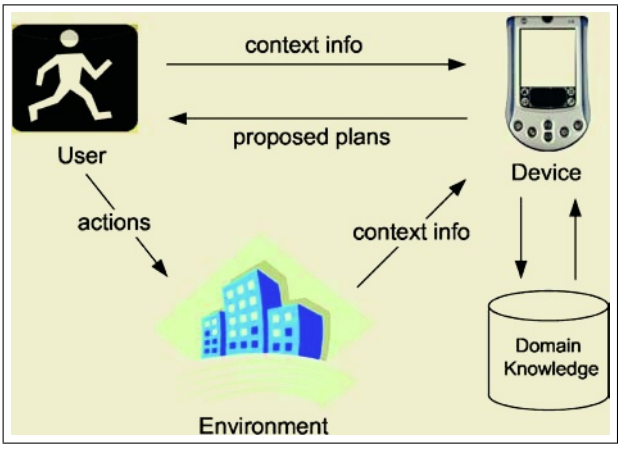

Fig. 1. Mobile device support for mobile users.

Mobility. The entire implementation must be adapted to the capabilities of mobile devices, taking into account their restrictive user interface, limited computational power and reliance on a finite power supply.

Practical Reasoning. Given a specified set of user goals, the system must be able to perform some form of planning in order to generate a set of actions that, if performed by the user, would achieve these goals.

Context-Sensitivity. Planning must take into account the current context in which the user is situated (e.g., the current user's physical location or the latest changes in the bus timetable). Information about contextual changes may be provided manually by the user or be automatically detected (e.g., changes in location could be tracked using a Global Positioning System (GPS) device).

Plan Selection. If there are a number of alternative plans for achieving the same goal, the system must be able to make a choice based on some comparison of the different plans. This may depend, for example, on the time needed, the overall cost, the risk factor, the user preferences etc. Appropriate decision mechanisms must therefore be supplied for supporting plan selection.

Plan Failure Recovery. If a plan fails at some stage, the system should be able to retract appropriately and select another alternative plan to suggest to the user. In order to allow for plan failure recovery, the interface must allow the user to easily indicate which parts of the plan have become unachievable and, possibly, for what reasons.

Conflict Resolution and Goal Selection. Sometimes, the user might have a number of goals that cannot be achieved simultaneously. In such cases, the system must be able to make a decision about which goals to try to achieve. In making such decisions, the system needs to take into account the importance of the goals as well as the costs of executing the plans.

In order to provide automated user support that has the above features, we propose an agent-based intelligent mobile assistant (AbIMA), running on a hand-held computer (e.g., a Palm Pilot). AbIMA will enable the user to enter information about the user's goals (which represent the high-level tasks, such as attending a lecture) as well as beliefs

about the world, the user's preferences, and so on. Then, AbIMA will reason about these goals, in the context of the beliefs it has acquired from the user and the environment, and provide the user with the appropriate plan of action.

In seeking a computational architecture that is suitable for providing the required functionalities described above, we found a significant overlap between the requirements of our domain and the concepts discussed in the belief-desire-intention (BDI) view of agency [3, 5, 18]. In particular, BDI-style agent architectures require explicit representations of agents' mental attitudes, such as beliefs, desires (or goals), plans, intentions (i.e. active plans), and so on. Various BDI-style architectures have provided different ways in which agents may reason about their mental attitudes in dynamic environments. Hence, BDI agent architectures seem to offer a promising basis for providing the features required in supporting mobile users[2].

In the context of our application, AbIMA will exploit the rich representation of BDI agents to capture information about Omar's beliefs, constraints, goals, preferences, plans, and so on. Then, AbIMA will use agent-based techniques to reason about these mental attitudes and make decisions about what plans (and subsequently, what actions) to suggest to the user. Another attractive notion of BDI architectures is that they are usually based on the aggregation of plans from plan libraries, encoded for a particular domain. This results in significant reduction of the complexities encountered in traditional planning systems, which is particularly helpful within the context of operation on limited mobile devices. Hence AbIMA will have a repository of pre-programmed plans that can be used and combined in order to achieve different goals. In particular, AbIMA is based on a specific BDI architecture and programming language called AgentSpeak(L) [17] [3], which we completely re-implement for running on mobile devices. In the subsequent section we discuss in more details the rationale behind the choice of AgentSpeak(L), as opposed to alternative BDI architectures.

2.3 Interaction between User, Environment, and Device

The interactions between user, device and the environment are illustrated in Figure 1. The device provides the user with suggested plans. In order to do so, the device uses its general domain knowledge, as well as dynamic contextual information about the environment and the user. The user then acts in the environment (e.g., moves from one place to another or completes a school assignment). These changes are observed by the device agent (either automatically or through user input), which might then make appropriate changes to the user's agenda.

Based on these interactions, the user and intelligent agent program can be viewed as offering each other complementary services based on their individual capabilities. The user and agent cooperate, each one of them dealing with those aspects of the problem to

[2] We concede that popular BDI architectures, and AgentSpeak(L) in particular, do not provide the details of *all* the features we need (such as mechanisms for reasoning about constraints and about conflicts between plans). Nonetheless, the BDI view offers a useful framework within which one may design these features and integrate them in a coherent manner.

[3] AgentSpeak(L) can be seen both as an agent architecture and an agent programming language. It is an architecture, in the sense that it has multiple interacting computational components, and a language, in the sense that it is executable from declarative program specifications.

which they are best suited. On the one hand, the agent has access to detailed, in-depth information about task plans, conflicting engagements and rules for dealing with conflict. On the other hand, the user provides the necessary contextual information based on the current environmental situation, his changing goals, and so on.

3 AgentSpeak(L) for Intelligent Assistance

In the previous section, we outlined the core requirements needed in AbIMA and briefly outlined why an automated assistant agent based on a BDI architecture has the potential to fulfil these requirements. In this section, we describe the AgentSpeak(L) programming language in some detail and show how some of the features needed in AbIMA may be implemented in AgentSpeak(L).

AgentSpeak(L) is an agent architecture/language with explicit representations of beliefs, goals and intentions. It was initially introduced by Rao [17], who provided a language for specifying BDI agents with a sketch of an abstract interpreter for programs written in the language. This, it was argued, provided the first bridge of the gap between BDI theory and practice. One reason we have chosen AgentSpeak(L), as opposed to other BDI models, is because it has an exact notation, thus providing an elegant specification of BDI agents. Moreover, AgentSpeak(L) has a clear, precise logical semantics, as well as being described in a computationally viable way [7]. This resulted in successful implementation of its abstract interpreter (as in [13], for example).

We now present a very brief overview of the AgentSpeak(L) syntax and its informal semantics. An AgentSpeak(L) agent is created by specifying a set of beliefs (called the *belief base*) and a set of plans (called the *plan library*). A *belief atom* is a predicate (as in Prolog). A *belief literal* is an atom or its negation. There are two types of goals in AgentSpeak(L). An *achievement goal* is a predicate prefixed with "!", stating that the agent wants to achieve a state of the world where the predicate is true. A *test goal*, on the other hand, is a predicate prefixed with "?" and states that the agent wants to test whether the associated predicate is a true belief (i.e., whether it can be matched with the agent's belief base).

A *triggering event* in AgentSpeak(L) is an event that causes a plan to be executed. There are two types of triggering events: events involving the addition "+" of a mental attitude (e.g., a belief or a goal) and events involving the deletion "-" of a mental attitude. Let us now explain how plans are represented [17, definition 5].

Definition 1. *(Plan) If* e *is a triggering event,* $b_1, ..., b_m$ *are belief literals, and* $h_1, ..., h_n$ *are goals or actions, then a plan is represented as follows*

$$e : b_1, \ldots, b_n \leftarrow h_1; \ldots; h_n$$

The expression on the left of the arrow is the head *of the plan, and the expression to the right of the arrow is the* body *of the plan. The expression* $b1, \ldots, bm$ *is referred to as the context in which the plan becomes* applicable.

Consider the AgentSpeak(L) plan below, expressing plan A.2 from section 2.1 above[4]:

[4] Note that actual plans can be more generic, using variables for time, locations etc. and retrieving those from knowledge sources such as local or network databases. For simplicity of presentation, however, we use instantiated plans.

Plan A.2
$+!location(user, faculty, 10am)$: $location(user, home, 8\_15am)$
$\leftarrow wakeup(user); !bus(home, city\_square);$
$!bus(city\_square, uni\_square); walk(uni\_square, uni\_rd)$

This plan states that in the event of adding a goal that the user be at the faculty at 10:00am, if the current context condition states that the user is at home and the current time is 8:15am, then the goal may be achieved by waking the user, taking a bus from to the City Square, taking another bus from the City Square to the University Square and finally walking from the University Square up the University Road. Recall that elements of the plan body may be directly executable actions (e.g., walking up the University Road) or may be sub-goals which themselves need plans to be achieved (e.g., taking a bus to the City Square involves going to the bus stop, boarding a particular bus etc.)

Note that in AgentSpeak(L), an event may be external or internal. An external event is one that is originated by a perception of the environment or from a given basic goal. In the above example plan, the goal $+!location(user, faculty, 10am)$ is an external event, since it is a goal provided by the user. An internal event, on the other hand, is one that is generated during the agent's process of plan execution. For example, in the course of executing the plan above, the second sub-goal of taking a bus from home to the City Square is fired as an internal event. This event may then trigger the plan described below, which corresponds to plan A.2.2.a described in section 2.1.

Plan A.2.2.a
$+!bus(home, city\_square)$: $location(user, home)$
$\leftarrow walk(home, teshreen\_rd); take\_bus\_number(12);$
$get\_off(city\_square)$

When an agent commits to a particular set of plans to achieve some goal, these partially instantiated plans (i.e., plans where some variables have taken values) are referred to as an intention associated with that goal. An AgentSpeak(L) agent can be described formally as follows [17, definition 6]:

Definition 2. *(Agent) An agent is a tuple*

$$\langle E, B, P, I, A, S_E, S_O, S_I \rangle$$

where E is a set of events, B is a set of base beliefs, P is a set of plans, I is a set of intentions, and A is a set of actions. The selection function S_E selects an event from the set E; the function S_O selects and option or applicable plan from a set of applicable plans; and S_I selects an intention from the set I.

We now explain the basics of the AgentSpeak(L) interpreter. At every cycle, AgentSpeak(L) updates the list of events. This causes relevant addition or removal of goals as well as possible updates to the belief base B. Updating beliefs must be done appropriately in order to ensure that B remains consistent. This may be done using some appropriate Belief Revision Function (BRF) [13]. The selection function S_E now selects an event from the list E. This event is unified against the triggering events at the heads of plans, generating the set of relevant plans. Now, for each plan, the context is tested against

the agent's beliefs. Those relevant plans that are unified successfully are the applicable plans. The function S_O now selects one of these plans. If the event was external, a new intention is created and placed in I. If, on the other hand, the event was an internal one, the selected plan is placed on top of the plan stack of an existing intention. Now, the function S_I selects an intention for execution. Recall that executing an intention may involve either executing direct actions or may involve triggering sub-goal events to be triggered. If the sub-goal is an achievement goal, it causes an internal event to fire; this may subsequently cause new plans to be added to the stack of plans associated with that intention (this happens in future reasoning cycles). If, on the other hand, the sub-goal is a test goal, the belief base is consulted to test whether the associated predicate is true. Actions and goals that are achieved are removed appropriately from the intention stacks. This ends a cycle of the interpreter. AgentSpeak(L) starts over again by checking the state of the environment and updating the belief base accordingly, generating events, and so on.

We now go through how some of the stages of the interpreter may be executed in our applications. In the context of an agent supporting a mobile user, an external event may be one of the following:

1. *User inputs goal.* At any stage, the user inputs the high-level goals he or she wishes to achieve. In our example, these would be represented by the tasks A, B, C and D described in the scenario above. These cause external achievement goal events of the form $+!goal(parameters)$ to be added to the set E. The user may also add test goals in the form of queries (e.g., the user may enquire about the current time, the location of a lecture theatre, and so on).
2. *User removes goal.* The user must also be allowed to remove goals, causing events of the form $-!goal(parameters)$ to be fired.
3. *User inputs or modifies belief.* The agent may perceive some new information about or changes in the environment. For example, the user may perceive that the buses from Teshreen Road to the City Square have been cancelled. In this case, the user may notify the agent of these changes by adding events of the form $+belief\_predicate(parameters)$ and $-belief\_predicate(parameters)$. Instead, the agent might be able to automatically retrieve such information, say from web services for bus timetables or from location positioning systems.

AbIMA may retrieve beliefs about Omar's current location, which is home, the current time, and so on. Omar would also inform the agent of its goal to reach the Faculty at 10:00am on a particular day by adding $+!location(user, faculty, 10am)$. After being selected by the event selection function S_E, this goal may be unified with the trigger event in the plan A.2. When the context condition of that plan is satisfied (i.e., when, based on the agent's clock and its beliefs about the user's location, predicate $location(user, home, 815am)$ is true in the belief base), the plan will become applicable. This plan will be compared with other possible applicable plans. If there are no other applicable plans at this stage, or if this plan is the most preferred, the selection function S_O will select this plan[5]. Since we are still at the top-level goal, a new intention will be generated and the plan will be added on top of the stack associated with that intention.

[5] This means that the user's preferences over alternative plans need to be encoded within the option selection function S_O itself.

Fig. 2. AbIMA introductory screen.

Now, the agent begins executing the plan. This involves informing the user about what actions he needs to take in order to achieve his goal. The action represented by the predicate $wakeup(user)$ is an atomic action and may be executed directly by activating the alarm on the Omar's handheld device. On the other hand, the activity of taking a bus to the City Square, represented in the predicate $bus(home, city_square)$, is not an atomic one; it is an achievement sub-goal that needs another plan to be executed before we can say it is achieved. This goal is posted as an internal goal, updating the set E of events. This event may then follow a similar process, triggering the plan A.2.2.a, and so on, until all activities in the plans reach the atomic level and hence can be executed directly, either by the agent or by Omar.

Note that during the agent lifecycle, unexpected events may also occur, such as a plan failing to be achieved due to the user's failure to achieve some basic action. For example, Omar may fail to take a particular bus due to road works. This requires a plan failure operator, which involves removing certain intentions, removing sub-goals, notifying the user, and so on. In this version, we do not fully deal with these issues[6].

4 Demonstrating AbIMA

In this section, we demonstrate the AbIMA user interface and explain how it supports the user through part of the scenario described in section 2.

We have implemented a complete version of the AgentSpeak(L) agent programming language on a Palm Pilot handheld device, running Palm OS, using Java 2 Micro Edition [10]. This forms the underlying engine of AbIMA.

Figure 2 shows the introductory screen of AbIMA. The user may choose to view and/or edit the beliefs, desires or intentions stored by AbIMA. In order to illustrate the

[6] Note that exactly how failure is dealt with was not fully articulated in Rao's original description of AgentSpeak(L) [17]. Therefore, we need to add a plan failure operator that is appropriate for our domain (this could be, for example, based on AgentSpeak(XL) [2]).

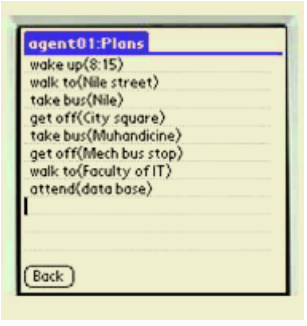

Fig. 3. Attend database lecture plan.

Fig. 4. AbIMA goals lists.

Fig. 5. Buy CD plan.

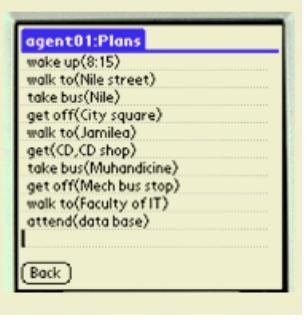

Fig. 6. Plan incorporating both user goals.

current functionality of AbIMA, we step through a simple example, which is based on, but slightly different to, the example presented in section 2.

By clicking on 'intentions', the user may view the current plans suggested by the agent. Suppose, in this case that the agent has generated a plan for attending a lecture on 'databases' at 9:30am. The result is shown in figure 3 [7]. The first action in the plan involves waking up at 8:15am. This is executed by the agent by activating the alarm of the PDA; the agent receives confirmation that this action has been executed after the user sets the alarm off (assuming the user is responsible and does not just go to sleep). Then, the agent advises the user to walk to the Nile Street, take the Nile bus, get off at the City Square, take the Muhandicine bus, get off at the 'Mech' bus stop and finally walk to the Faculty of IT. It is possible to include 'user interface plans' based on the current context. For example, when the user reaches the City Square, the agent may display a map of the area indicating the location of the Muhandicine bus stop.

By clicking on 'desires', the user may add a new goal. Suppose the user would like to add a goal of getting a CD from the CD Shop. This means there are now two goals (see Figure 4). In an initial version of AbIMA, the agent generates a plan for getting the CDs

[7] Currently, AbIMA presents suggested plans as stand-alone lists. One could, however, integrate this list into an existing calendar application.

given the current context (i.e., given the user is still at home). This results in the plan in figure 5. However, note that this plan also involves going to the City Square. In this case, there is an opportunity for the agent to combine the two plans so as to achieve both goals. This would involve advising the user to pass from the CD shop, while on the way to the lecture. In order to enable AbIMA to perform such reasoning we introduce the notion of *expected beliefs*. These *expected beliefs* are the beliefs that the agent would *expect* to hold true once the second plan (Buy CD) has been achieved. Given these beliefs the agent attempt to find a plan for the first goal (Attend database lecture) from this new context. If a plan can be found that corresponds to a state that the agent would find itself in any case (i.e. beginning from the City Square) the two plans are merged to produce a common plan for both goals. This allows AbIMA to recognise the opportunity to get the CD while in the city. AbIMA hence generates the plan shown in figure 6 [8]. Such a solution is certainly not general and can only work under the limited context of common locations and time restrictions, but proves effective in a wide variety of settings, which indicate that it represents a worthwhile avenue for further investigation.

5 Implementation Challenges

During our work on AbIMA, we have identified a number of important challenges. While the use of the AgentSpeak(L) architecture and our implementation of it goes some way towards addressing some of the issues, others have been dealt with in an ad hoc manner. Nevertheless, each one of them provides fertile ground for further research. We discuss the most salient of these challenges below.

Limited Storage. Handheld devices often have limited storage capacity. The agent might need comprehensive timetable information, maps to direct the user, images identifying landmarks to assist the user in establishing his current position, and so on. This creates the need for allowing information modules to be plugged into and out of the device. This could be done through synchronisation with the PC or using a wireless connection. The agent may download these information modules on demand based on the upcoming tasks.

Limited Processing Power. Similarly, due to the limited processing power of mobile devices, special care needs to be taken to ensure proper execution. For example, appropriate design needs to ensure that no extremely long intention stacks are accumulated. Also, the user might need timely response in certain circumstances. This could be facilitated, for example, by producing suggested actions to the user before the complete agent cycle is completed. To avoid these issues, we currently use relatively small plan libraries, belief bases etc.

Plan Configurability and Interoperability. Due to the limited storage and computational power of handheld devices, the agent architecture should allow for configurable plan libraries that allow the user to supply the agent with the right plans to deal with the upcoming tasks. Moreover, users may wish to download new plans or share plans with other users. One possible solution is to express plans in a shared format based on, for example, the extensible markup language XML.

[8] We shall provide the details of this solution in a future paper.

Online/Offline State Preservation. When users are mobile while engaged in their tasks, they cannot be expected to have the mobile device running all the time. They might also use different devices. This raises the problem of preserving the agent's state properly on the device. This state may also be represented in a modular fashion in order to be backed-up, transported unto a different machine (e.g., desktop computer), and so on.

Perception/Action Representation. Initially, AgentSpeak(L) was designed with a view to implement agents acting autonomously in an environment, such as a robot in a factory cell. In the case of supporting mobile users, on the other hand, additional considerations, relating to the way the agent perceives the environment and acts upon it, must be catered for. For example, we cannot expect the user to have the handheld computer switched on continuously; hence, there is no continuous flow of action and perception. Similarly, the limited input capabilities of mobile devices and the limited availability of mobile users (e.g., due to their engagement in the task) may impede the agent's ability to have up to date, detailed information about the environment.

Situation Awareness. An important challenge related to supporting mobile users is that of modelling situation awareness. How can the agent establish an up-to-date perception of its environment? To address this problem, we have data structures explicitly representing time, tasks, goals, locations etc. and we allow the user to view and manipulate these structures directly. These data structures can also be updated based on other sources than the user, such as bus timetable databases etc. Other relevant questions include: If multiple users were connected, how could they establish joint awareness of their corresponding situations? If the user is allowed to switch between different platforms, how is situation awareness affected [19]? These issues are outside the scope of our current work.

As mentioned above, the challenges related to the limited storage and computational capacity of handheld devices may be dealt with by providing a modular, configurable implementation of the AgentSpeak(L) architecture. Therefore, we took a component-based approach to our design with a clear separation between information modules and decision-making modules so that they can be easily changed.

6 Related Work

One related project is the Electric Elves [4], in which multiple agents support each human (e.g., the mobile device, fax machines and desktop computer, each has its own agent) and agents interact to assist people coordinate their activities. The Electric Elves project builds on an ad hoc agent architecture that integrates different artificial intelligence technologies for knowledge representation, planning, teamwork etc. Our work, on the other hand, focuses on the use of a particular agent programming language and investigates its applicability in the domain of mobile user support. We hope that using a formal agent language such as AgentSpeak(L) would later allow us to formally verify some of the properties of mobile assistant agents, such as adjustable autonomy (where decision making control shifts between the user and the agent), failure recovery, and so on. In this sense, our work is complementary to the Electric Elves project since it permits studying the issue of control flow within a well-defined agent language context.

Another related body of work is the NurseBot project [15]. This project is concerned with the construction of a robot capable of providing personal assistance to the elderly. The robot is capable of performing tasks such as reminding the human of things to be done, helping the human move from one room to another, manipulating objects (e.g., fridge, laundry), and so on. While the application domain of the NursesBot project is different from AbIMA, they share many challenges. For example, both systems require capabilities of planning and reasoning about context. On the technical level, AbIMA differs from NurseBot in that, whereas we are proposing to use the BDI architecture as a foundation, NurseBot uses techniques from AI planning and Bayesian reasoning.

Finally, recall that AbIMA needs to plan ahead while making sure plans do not change the context in such a way as to make future plans inapplicable. Some work on detecting and resolving conflicts between plans in BDI agents is presented by Thangarajah et. al. [20]. To achieve the same end, the AgentSpeak(XL) programming language [2] integrates AgentSpeak(L) with the TAEMS scheduler [6]. Moreover, related theoretical work on selecting new plans in the context of existing plans is presented in [9]. We are currently investigating whether such work could be effectively implemented in our domain, given the limitations of mobile devices.

7 Conclusion

In this paper, we presented AbIMA, an agent-based intelligent mobile assistant, running on a hand-held device, for supporting users prior to and during the execution of their tasks. We presented a scenario involving a student executing a number of tasks on campus. We used this scenario to extract the essential features required in an automated agent assisting the user during task execution. We then argued that the belief, desire, intention model of agency seems to cater for these features. AbIMA's implementation is based on the well-known agent architecture and programming language AgentSpeak(L), which provides explicit representations of agents' beliefs, desires and intentions, as well as an abstract interpreter for reasoning about these mental attitudes to guide the performance of actions. We described how the features required in AbIMA can be specified in AgentSpeak(L). Finally, we outlined some technical challenges we faced, hence motivating future work in the area.

While AgentSpeak(L) seems to provide the necessary features required for our application requirements, a number of domain specific issues arise. The agent's perception and action are done through a user engaged in a task. A required action could be performed directly by the device (e.g., activating the wake-up alarm) or indirectly by instructing the user to perform some activity (e.g., to take a particular service bus). The two types of actions must be clearly differentiated and a comprehensive study of the control flow between the user and agent are needed.

Another future direction would be connecting the device into a network of mobile assistants. This raises at least two families of issues. Firstly, there are issues related to teamwork, coordination and negotiation in multi-agent systems. Mobile assistants belonging to different users may interact, for example, to coordinate a meeting or negotiate a restaurant for a group dinner that satisfies the users' dietary requirements. Secondly, there are technical issues related to, for example, representation and retrieval of planning information and other information from different sources.

Acknowledgements

The authors are very grateful to Rafael Bordini and the anonymous reviewers for their very valuable comments that helped improve this version of the paper. During this work, Iyad Rahwan was supported by a Melbourne University Research Scholarship and a CMIS–CSIRO Ph.D. Top-Up Scholarship.

References

1. F. Bergenti, A. Poggi, B. Burg, and G. Claire. Deploying FIPA-Compliant Systems on Handheld Devices. *IEEE Internet Computing*, 5(4):20–25, 2001.
2. R. H. Bordini, A. L. C. Bazzan, R. O. Jannone, D. M. Basso, R. M. Vicari, and V. R. Lesser. AgentSpeak(XL): Efficient intention selection in BDI agents via decision-theoretic task scheduling. In C. Castelfranchi and W. L. Johnson, editors, *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002)*, pages 1294–1302, New York, USA, 2002. ACM Press.
3. M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, Cambridge MA, USA, 1987.
4. H. Chalupsky, Y. Gil, C. A. Knoblock, K. Lerman, J. Oh, D. V. Pynadath, T. A. Russ, and M. Tambe. Electric elves: Applying agent technology to support human organizations. In H. Hirsh and S. Chien, editors, *Proceedings of the 13th International Conference of Innovative Application of Artificial Intelligence (IAAI-2001)*. AAAI Press, 2001.
5. P. Cohen and H. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
6. K. S. Decker and V. R. Lesser. Quantitative modeling of complex environments. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 2(4):215–234, 1993.
7. M. d'Inverno and M. Luck. Engineering AgentSpeak(L): A Formal Computational Model. *Journal of Logic and Computation*, 8(3):233–260, 1998.
8. C. Heinze and S. Goss. Human performance modelling in a BDI system. In *Proceedings of the Australian Computer Human Interaction Conference*, 2000.
9. J. F. Horty and M. E. Pollack. Evaluating new options in the context of existing plans. *Artificial Intelligence*, 127(2):199–220, 2001.
10. Java 2 Micro Edition. `http://java.sun.com/j2me/`
11. N. R. Jennings and M. J. Wooldridge. Applications of intelligent agents. In N. R. Jennings and M. J. Wooldridge, editors, *Agent Technology: Foundations, Applications, and Markets*, pages 3–28. Springer-Verlag, Heidelberg, Germany, 1998.
12. Z. Maamar, W. Mansoor, and Q. H. Mahmoud. Software agents to support mobile services. In C. Castelfranchi and W. L. Johnson, editors, *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002)*, pages 666–667, New York, USA, 2002. ACM Press.
13. R. Machado and R. H. Bordini. Running AgentSpeak(L) agents on SIM_AGENT. In J.-J. Meyer and M. Tambe, editors, *Pre-proceedings of the 8th International Workshop on Agent Theories, Architectures and Languages*, volume 2333 of *Lecture Notes in Computer Science*, Berlin, Germany, 2002. Springer Verlag.
14. H. Nwana, D. Ndumu, D. Lee, and J. Collis. ZEUS: A toolkit for building distribtued multi-agent systems. *Applied Artificial Intelligence*, 13(1):129–186, 1999.
15. M. E. Pollack. Planning technology for intelligent cognitive orthotics. In *Proceedings of the 6th International Conference on AI Planning and Scheduling*, 2002.

16. T. Rahwan, T. Rahwan, I. Rahwan, and R. Ashri. Towards a mobile intelligent assistant: Agentspeak(l) agents on mobile devices. In P. Giorgini and M. Winikoff, editors, *Proceedings of the 5th International Bi-Conference Workshop on Agent Oriented Information Systems (AOIS)*, 2003.
17. A. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. V. de Velde and J. W. Perram, editors, *Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, volume 1038 of *LNAI*. Springer, 1996.
18. A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First International Conference on Multiagent Systems*, San Francisco, USA, 1995.
19. J. C. Tang, N. Yankelovich, J. Begole, M. V. Kleek, F. Li, and J. Bhalodia. ConNexus to awarenex: extending awareness to mobile users. In *Proceedings of the SIGCHI conference on Human Factos in Computing Systems*, pages 221–228, 2001.
20. J. Thangarajah, L. Padhgam, and M. Winikoff. Detecting and avoiding interference between goals in intelligent agents. In G. Gottlob and T. Walsh, editors, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2003)*. Academic Press, 2003.
21. M. J. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.
22. A. Zaslavsky and Z. Tari. Mobile computing: Overview and current status. *Australian Computer Journal*, 30(2):42–52, 1998.

Market-Based Recommendations: Design, Simulation and Evaluation*

Yan Zheng Wei, Luc Moreau, and Nicholas R. Jennings

Intelligence, Agents, Multimedia Group
School of Electronics and Computer Science
University of Southampton, UK
{yzw01r,L.Moreau,nrj}@ecs.soton.ac.uk

Abstract. This paper reports on the design, implementation and evaluation of a market-based recommender system that suggests relevant documents to users. The key feature of the system is the use of market mechanisms to shortlist recommendations in decreasing order of user-perceived quality. Essentially, the marketplace gives recommending agents the incentive to adjust their bids to different levels according to their belief about the corresponding user-perceived quality. In order to test the efficiency of our marketplace design, this paper reports on our simulation results for different types of users with different information needs. In this context, we demonstrate that the bids from recommendations with different user-perceived quality levels converge at different price levels and that the bidding agents can relate their bids to their internal belief about the quality of their recommendations.

1 Introduction

This paper reports on the design, implementation and evaluation of a recommender system that seeks to manage the problem of information overload. Generally speaking, such systems help make choices among recommendations from all kinds of sources without having sufficient personal experience of all these alternatives [1]. In a typical system, recommendations are provided as inputs and the system then aggregates and directs them to appropriate recipients. Thus, a recommender system's main value lies in information aggregation and its ability to match the recommenders with those seeking recommendations.

Recommender systems have been applied to many application domains and many different techniques have been used to make the recommendations. For example, some are based on the correlation between the item contents (such as term frequency inverse document frequency [2] and weighting [3]), while others are based on the correlation between users' interests (such as votes [4] and trails [5]). However, there is no universally best method for all users in all situations [6,7] and we believe this situation is likely to continue as ever more

* This research is funded in part by QinetiQ and the EPSRC Magnitude project (reference GR/N35816).

P. Giorgini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 61–77, 2004.

methods are developed. Moreover, the ranking of relevance produced by the different methods can vary dramatically from one another. Given this situation, we believe the best way forward in this area is to allow the multiple recommendation methods to co-exist and to provide an overarching system that coordinates their outputs such that only the best recommendations (from whatever source or method) are presented to the user. To this end, in [8] we developed a system that integrates multiple recommendation methods to help the users with the problem of "where to go next?" when they browse the Web (see Fig. 1). This system used a *market-based approach* to achieve such coordination. This is because the problem of selecting appropriate recommendations to place in the limited sidebar space can be viewed as one of scarce resource allocation and markets are an effective solution for this class of problems [9]. This market structure was refined in [10] to one that theoretically is Pareto-optimal, social welfare maximizing, stable and fair to all the recommending agents. Given this analysis, this paper reports on the actual implementation of the marketplace and its evaluation to determine whether the theoretical properties hold in practice. This, in turn, is the key to whether the market can actually be used as an overarching coordination framework for our practical software development endeavour (which is the next stage of our work). In more detail, we develop three kinds of agents (reward agents, recommending agents and user agents) that operate in the marketplace and a number of simulations were performed to assess the system's operational characteristics.

Fig. 1. Browser with Recommendations.

The remainder of this paper is structured in the following manner. Section 2 further develops the market mechanism in [10] and details the behaviour of the three kinds of agents. Section 3 details and then empirically evaluates the performance of the marketplace. Section 4 outlines related work in terms of reducing

information overload and market-based systems. Finally, section 5 concludes and points to future work.

2 Designing the System

Our marketplace operates according to the following metaphor. A *user agent* acting on behalf of the user is selling sidebar space where recommendations may be displayed (see left hand side of Fig. 1). The number of such slots is fixed and limited. Information providers (the component *recommending agents*) want to get their recommendations advertised in the user's browser and so compete in the marketplace to maximize their individual gain by purchasing this advertising space when they have what they believe are good recommendations. Their bids indicate how much they are willing to pay for such slots. The recommender system acts as the auctioneer and selects the most valuable items (highest bids) which it then displays as its recommendations (those agents that provided these *shortlisted* items are then charged according to their bids). The user then chooses some of these recommendations (or not) according to their interests. The agents that provided the user-selected recommendations receive some reward (since these recommendations are deemed useful) from the *reward agent*, while those not chosen receive no reward. The agents with recommendations rewarded might increase their revenue and those agents that make poor recommendations make losses (since they have to pay to advertise their recommendations). Thus, over the longer term, the agents that make good recommendations become richer and so are able to get their recommendations advertised more frequently than those whose recommendations are infrequently chosen by the user.

In [10] we introduced the valuation of a recommendation from two perspectives. Firstly, from the viewpoint of the user, how well a recommendation satisfies the user is termed the *user-perceived quality*. Secondly, from the viewpoint of a recommending agent with a specific recommendation method, the relevance score it computes for a particular recommendation is termed its *internal quality*. The role of the marketplace is then to try and connect these two quality values by imposing a reward regime that incentivises the recommending agents to bid in a manner that establishes an appropriate correlation between these values and their bid price. To evaluate our mechanism we will use the following evaluation metrics:

Price Convergence: This happens if the price with respect to a user-perceived quality level converges after a number of rounds of auctions. This is important from the viewpoint of the bidding agents since it enables them to learn to bid rationally at a certain level to maximize their revenue. Without convergence, an agent will never know how much to bid with respect to a given recommendation and therefore the marketplace behaviour will be chaotic.

Efficient Shortlists: The marketplace should be capable of shortlisting the recommendations in decreasing order of the user-perceived quality after a number of auction iterations. This is important from the point of view of the users since they only want a small number of the best recommendations.

Clear Incentive: The protocol should be able to incentivise the recommending agents to bid differently for different internal quality levels. This is important because the agents need to relate their bids to the internal quality of the recommendations through the feedback from the marketplace which reflects the user's preference.

Stability: A protocol is stable if it provides all agents with an incentive to behave in a particular way over time. The marketplace should be designed to be stable because, if a self-interested agent is better off behaving in some other manner than desired, it will do so. Therefore, stability is important because without it the system behaviour is unpredictable.

Fairness: In our context, a protocol is fair if it gives all recommendations equal opportunity of being shortlisted (irrespective of the agent or method that generates them). This is important because we want the system to shortlist the best recommendations in an unbiased manner.

With these metrics in place, we first outline the auction protocol for our marketplace (see [10] for more details and a justification of our design choice) before detailing the recommending, the user and the reward agents (see Fig. 2). The particular protocol we employ is a *generalized first-price sealed-bid auction*, in which all agents whose recommendations are shortlisted pay an amount equal to their valuation of the advertisement and those whose recommendations selected by the user are awarded. In more detail, the market operates in the following manner. Each time the user browses a new page, the auction is activated. In each such activation, the auctioneer agent calls for a number of bids (one bid contains one recommendation and its corresponding price), say M ($M > 0$ and the value of M is fixed) equal to the number of recommendations it is seeking. There are S recommending agents in the system and each recommending agent submits at most M bids in each auction. After a fixed time, the auctioneer agent ranks all the bids it has received by their bidding price and directs the M bids with the highest prices to the user's browser. Those agents with shortlisted bids pay according to how much they bid. Those agents whose recommendations are non-shortlisted pay nothing. The user may then take up a number of these shortlisted recommendations in which case the agents that supplied them are rewarded.

2.1 Designing the Reward Agent

The reward agent determines the reward paid to the agents who make recommendations selected by the user. Given a fixed amount of total payoff to be distributed to N user selected recommendations, we have designed a Pareto efficient and social welfare maximizing reward mechanism [10]. The reward mechanism is as follows. All user-selected recommendations are ordered by the reward agent in decreasing rank of user-perceived quality (such that $Q_1 > Q_2 > \cdots > Q_N$). The user-selected-recommendation with the h^{th} highest user-perceived quality, Q_h ($h \in [1..N]$), is termed *the h^{th} rewarded recommendation*. To simplify the description of the reward mechanism, the following variables are used: let P_{M+1}

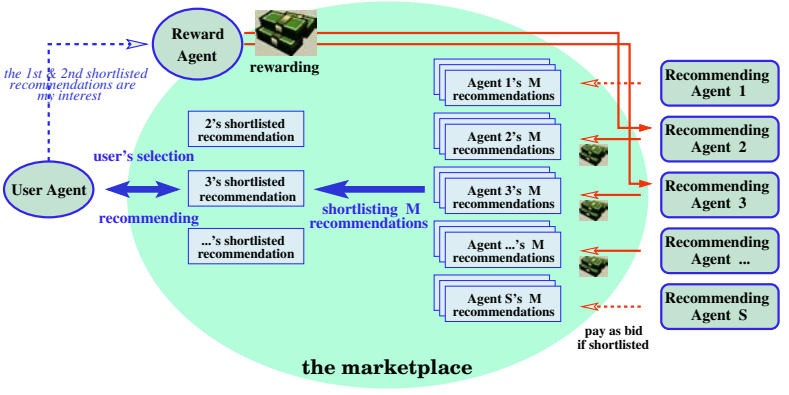

Fig. 2. The Auction Protocol

be the highest non-shortlisted bid; P_h be the bidding price of the h^{th} rewarded recommendation; and P_m^* $(m \in [1..M])$ be the historical average bidding price of the m^{th} shortlisted recommendation during the system's lifetime (note the bidding agents do not actually know this value). With these definitions in place, the reward to the h^{th} rewarded recommendation, R_h, is defined as:

$$R_h = \delta \cdot Q_h \cdot P_{M+1} - \alpha \cdot |P_h^* - P_h| \tag{1}$$

where δ and α are two system coefficients ($\delta > 0$ and $\alpha > 1$) that are set according to the system's desired characteristics (see section 3).

2.2 Designing the Recommending Agents

In this subsection, we discuss how an agent generates a bid and how it relates the bidding price to its internal quality for a recommendation.

Each agent has a set of recommendations available to suggest. From the point view of a recommending agent, what it needs to do is to compute the relation between its local perception of relevance and the user's current context. Having done this, it can then bid an appropriate price to maximize its revenue. Given a specific recommending agent, the agent will relate its bidding price to its knowledge about the user-perceived quality (reflected by the rewards it has received) with respect to different internal quality levels. We term this relationship between the bidding price and the internal quality an agent's *strategy profile*. This profile is on a per agent basis. It records an agent's bidding price for different internal quality levels and indicates how an agent should relate its bid to its internal quality.

In order to build up its strategy profile, an agent will divide its range of internal quality values into a number of equal segments. For each such segment, the agent will base its next bid (P^{next}) on its last bid price (P^{last}) and what happened to this bid:

1. **The Bid Was Not Shortlisted**
   The agent will increase the bid price to get them shortlisted:
   $P^{next} = Y \cdot P^{last} \quad (Y > 1)$.
2. **The Bid Was Shortlisted But Not Rewarded**
   The agent will decrease the bid price to lose less:
   $P^{next} = Z \cdot P^{last} \quad (0 < Z < 1)$.
3. **The Bid Was Rewarded**
   In [10] we proved that an agent can be aware of whether its bid price is close to or far from the market equilibrium price with respect to the user-perceived quality of the segment. This can be achieved from the agent's knowledge about the *profit* (reward obtained minus bidding price that has been paid with respect to one recommendation) made from the segment in the last few rewarded rounds. The fact is that the profit increases (or decreases) when the bid price is close to (or far from) the equilibrium price. We also proved that only by minimizing the difference between its bids and the equilibrium price ($|P_h^* - P_h|$ see equation (1)) can an agent maximize its revenue. Therefore, an agent will keep increasing or decreasing the next bid price until there is a profit loss has occurred and then adjust its price inversely. To describe the behaviour with respect to the last bid rewarded, several variables are involved: *current profit* is the profit made from the current auction and *last profit* is that of the last auction; ΔP is the small amount of adjustment made to the last bid so as to move closer to the equilibrium price (each time ΔP changes its sign, its absolute value shrinks to chase the equilibrium price more accurately with respect to the user-perceived quality of the segment); P_{init} is the initial value; P_{lim} is the minimum absolute value of ΔP ($|\Delta P|$ cannot shrink indefinitely because otherwise it will reduce to 0 and would become useless). With these variables in place, the behaviour with respect to last bid rewarded is defined as below:

$\Delta P = P_{init}$ // *initialize the price adjustment*
IF (*current profit* > *last profit*) // *there is no profit loss*
    $P^{next} = P^{last} + \Delta P$ // *increase or decrease bid price*
ELSE // *there is a profit loss*
    $\Delta P = (-1) \cdot \Delta P$ // *change the sign of the adjustment*
    IF ($|\Delta P| > P_{lim}$)
        $\Delta P = X \cdot \Delta P \quad (0 < X < 1)$ // *shrink the adjustment*
    ELSE
        $\Delta P = \frac{\Delta P}{|\Delta P|} \cdot P_{init}$ // *enlarge to the initial value*
    $P^{next} = P^{last} + \Delta P$ // *increase or decrease bid price*

2.3 Designing the User Agent

To evaluate our marketplace we need to simulate the choices of a user in selecting recommendations. We do this by deploying a user model inside the user agent. Building on the user simulation of [11], we adopt the following models. When a user faces a set of shortlisted recommendations, the user will first visit some of

Table 1. User's Decision of Different models.

Shortlisted Recommendations	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9	Q_{10}
User Perceived Quality	70	50	75	30	60	82	90	85	65	55
Decision of Independent Selection	1	0	1	0	1	1	1	1	1	0
Decision of Search-Till-Satisfied	1	0	1	0	1	1	0	0	0	0

Both models have the same AT of 60. Search-till-satisfied model has a ST of 80. "1" means the recommendation is selected to be rewarded, while "0" means not selected.

them and will then have a valuation of each visited recommendation. Thus, a user assigns a number, Q_i ($i \in [1..M]$, $Q_i \in [0..100]$), to each visited recommendation according to his/her evaluation of the recommendation. This number Q_i corresponds to the user-perceived quality. Based on this value, a user's behaviour with respect to selecting shortlisted recommendations can be classified by:

Independent Selection: The selection of one recommendation is independent of the others. Once the user-perceived quality of a recommendation is greater than or equal to a particular *acceptance threshold* (AT), the recommendation is accepted and rewarded. Those recommendations with user-perceived quality less than AT will not be selected and therefore receive no reward.

Search-Till-Satisfied Behaviour: The selection of one recommendation is dependent on other recommendations that are ranked above it in the list. In this case, the user stops searching when a recommendation is encountered that has a user-perceived quality greater than or equal to a particular *satisfaction threshold* (ST).

By means of an illustration, Table 1 is an example of a user's decision under the two different models. All recommendations with user-perceived quality above the AT (60) are selected in case of independent selection. However, Q_7, Q_8 and Q_9 are not selected by the search-till-satisfied behaviour though their user-perceived quality is above the AT. This is because Q_6 with a quality of 82 is found and since this is above the ST (80) the user stops searching.

3 Simulation and Evaluation

We have simulated the marketplace in order to study its properties and its suitability for producing good recommendations. To this end, this section presents the experimental settings of the simulation with respect to the reward agent, the recommending agents and the user agent. The system properties are also evaluated empirically.

3.1 Experimental Setting

The configuration of the reward agent, recommending agents and user agent are presented in this subsection. The number of bids called for (M) is not under the control of any of these three kinds of agents and we set it to the value of ten.

Reward Agent. The reward agent controls two system variables: δ and α (see equation (1)). δ affects the volume of the credit paid to a user-selected-recommendation. The bigger δ is, the more the recommendation is paid. α affects the sensibility of the incentives the marketplace delivers to the recommending agents to let them be aware of the equilibrium (because the recommending agents need large alterations to chase the equilibrium price if α is big). In our experiment, we set $\delta = 1.5$ and $\alpha = 1.5$ based on our experience that these values enable the recommending agents to increase their revenue by making good recommendations over the long term and chase the equilibrium quickly.

Recommending Agents. Our marketplace contains many different recommendation methods (represented as agents) that will each have their own valuation of the relevance of a particular recommendation to the user in a particular context. This means the same document may have varying internal quality scores with each different method. The next step is to determine how to simulate the internal qualities of different recommendation methods. Based on our previous studies in this area, by randomly collecting 400 different Web pages on the subject of "world news", we find that the keyword similarity [8] of the 400 documents compared to the CNN's frontpage (`http://www.cnn.com`) is likely to follow a Guassian normal distribution (see the contour of the distribution in Fig. 3). In our experiments, therefore, we decide to simulate the internal quality of one method by a random variable that follows a Guassian distribution. The probability density function of the Guassian distribution is defined as[1]:

$$N(\mu, \sigma^2): \quad f(x) = \frac{1}{\sqrt{2\pi}\ \sigma}\ e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \qquad x \in [0, 1]$$

where μ and σ are the mean value and the standard deviation of the random samples. We assume that three recommendation methods are used in our system. Each recommendation method relates to one of the three distributions: $N(0.35, 0.1^2)$, $N(0.5, 0.1^2)$ and $N(0.65, 0.1^2)$ (see Fig. 4). These distributions were chosen based on our previous studies [8] in this area. In this case, the mean of one distribution represents the average value of the internal quality of all samples generated by the corresponding method. The middle range between one unit of the standard deviation on both sides of the mean of one distribution contains the majority of the samples (about 68 percent of its total). We set the standard deviation to a small value (0.1). This means the three different methods share only a few internal quality levels (this is important because otherwise it is hard to distinguish different methods).

One of the key objectives of the recommending agents is to build up their strategy profiles so that they can relate their bidding price to their internal quality based on their knowledge about the reward (which reflects the user-perceived quality of the recommendations). In order to learn such characteristics for all internal quality levels, each agent segments its strategy profile into 20 continuous segments. In each auction, a recommending agent needs to compute the

[1] We fix the sample into the range [0, 1] (rather than (-∞, +∞)) because most internal quality results are a percentage number and are positive.

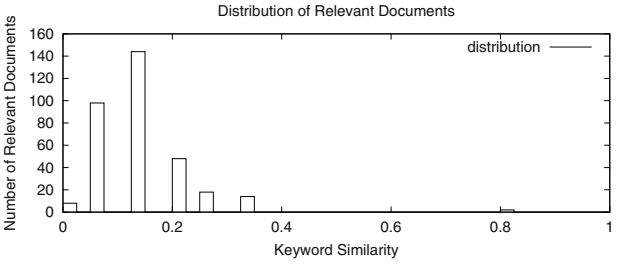

Fig. 3. The Contour of the Relevant Documents Distribution.

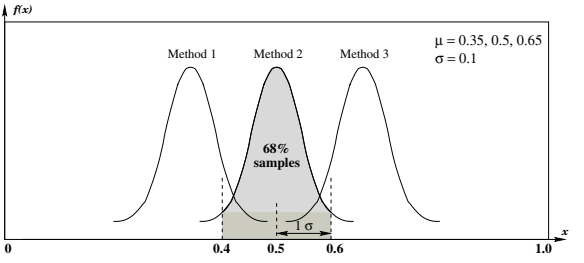

Fig. 4. Different Recommendation Methods' Internal Quality Profiles.

internal quality of ten recommendations and make ten corresponding bids. In the early auction rounds, all agents' strategy profiles are empty. With an empty strategy profile, an agent will bid proportionally (because an agent can only expect a high internal quality recommendation to receive a high user-perceived quality and more reward than a low internal quality recommendation) to the internal quality of ten recommendations based on an initial seeding price. We set different values (randomly generated from the range [128, 256]) for different recommending agents (because different agents evaluate their recommendations differently with their empty strategy profiles). After each auction, all strategy profile segments that any of the ten bids belong to record and update information about: the last bid status (not shortlisted, shortlisted but not rewarded or rewarded), the last bid price, the last rewarded price, the last rewarded profit, ΔP, and the number of rewards. Based on such information about each segment, and using the bidding strategy presented in subsection 2.2, an agent can compute its bids in subsequent auctions if there are recommendations that belong to this segment. After a number of iterations, those segments that cover the majority of samples will have sufficient information to reach the equilibrium price and form a stable strategy profile.

User Agent. When real users use a recommender system, they actually visit the shortlisted recommendations (or not) and therefore have a valuation about how much each recommendation satisfies them. Hence, when simulating the user we assume the user agent knows its valuation for each recommendation and

correspondingly we assign the user-perceived quality based on this valuation. From a recommending agent's viewpoint, the internal quality reflects the internal valuation of a recommendation. However, the relationship between the user-perceived quality and the internal quality of any given recommendation is hard to determine. To this end, in simulating user-perceived quality for a given recommendation we make two assumptions[2]:

Assumption 1: *With respect to any given recommendation method (represented as $N(\mu, \sigma^2)$), there exists a relationship* $\mathbf{f}$ *between the user-perceived quality Q and the internal quality q of a recommendation.* More formally:

$$\forall N(\mu, \sigma^2),\ \exists \mathbf{f} \quad \longrightarrow \quad \textit{if}\ q \sim N(\mu, \sigma^2),\ \textit{then}\ Q = \mathbf{f}(q)$$

Assumption 2: *Based on assumption 1, given a recommendation method that follows a specific internal quality distribution $N(\mu, \sigma^2)$, a recommendation with higher internal quality q receives higher user-perceived quality Q.* To simplify the problem, we further assume that Q has a linear relationship $\mathbf{f}$ with q with respect to a given $N(\mu, \sigma^2)$. More formally:

$$\forall N(\mu, \sigma^2),\ \exists \mathbf{f} \quad \longrightarrow \quad \textit{if}\ q \sim N(\mu, \sigma^2),\ \textit{then}\ Q = \mathbf{f}(q) = k \cdot q + b \quad (k > 0)$$

where b is a coefficient that can be any value.

Based on these assumptions, when faced with recommendations provided by different methods (represented by different $N(\mu, \sigma^2)$), a user has a different value of k and b to assign to the value of Q for q generated from $N(\mu, \sigma^2)$. With respect to the three recommendation methods of Fig. 4, what a user agent does is to normalize all the internal quality samples generated from the three distribution functions into one with a single distribution. To simplify this process, we define the normalization function as follows:

$$Q(q) = \begin{cases} 100 \cdot (q + 0.15) + \theta & \text{if } q \sim N(0.35, 0.1^2) \\ 100 \cdot q + \theta & \text{if } q \sim N(0.50, 0.1^2) \\ 100 \cdot (q - 0.15) + \theta & \text{if } q \sim N(0.65, 0.1^2) \end{cases} \tag{2}$$

where θ is the noise added to q and is, in fact, another random variable that follows a uniform distribution within the range of [-5, 5]. Thus, the value of Q is within the range of [0, 100] irrespective of the noise.

3.2 Analysis

Having outlined the set up of the three kinds of agents specified in subsection 3.1, this section evaluates the system properties, through various simulations, with respect to the evaluation criteria specified in section 2.

Price Convergence. In [10], we proved that the marketplace can reach an equilibrium such that the shortlisted prices converge at different levels with respect to

[2] We believe both of these assumptions are reasonable as an initial point of departure. In the future, however, we seek to assess their validity in practice.

different user-perceived quality levels. To evaluate this, we arranged 200 auctions with 10 shortlisted recommendations using the independent selection user model ($AT = 58$) and 9 recommending agents (every three share one method defined in Fig. 4) to see if the marketplace does indeed have such a convergence property. From Fig. 5, we can see that the shortlisted prices converge (for example, the 4^{th} and 10^{th} bid oscillate around 180 and 170 respectively, which indicate P_4^* and P_{10}^* respectively) after about 60 auctions. Because of the limited space, we do not show the case of the search-till-satisfied user model being used in a separate figure (the market converges slowly with $ST = 80$ and $AT = 58$ compared with the independent selection since fewer agents are rewarded in this case and they need more bids to chase the equilibrium price). From our simulations, we find that the shortlisted prices always converge after a number of iterations but the speed of the convergence depends on the setting of the parameters α, AT, ST, Y, Z and ΔP (as introduced in subsection 3.1). Because of the limited space, we do not discuss all the effects of these variables in detail[3].

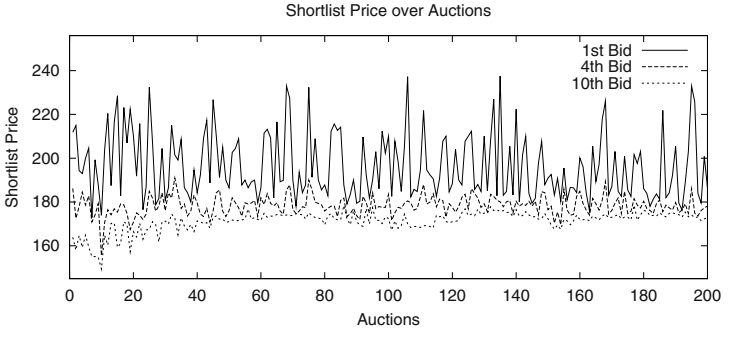

Fig. 5. Convergence of Shortlisted Prices.

Efficient Shortlists. The most important feature of our system is its capability to shortlist the best recommendations in decreasing order of user-perceived quality when the market converges. To this end, Fig. 6(a) shows the user-perceived quality of the shortlisted recommendations at the 60^{th} auction (which is after the convergence). We can see that the quality of the ten shortlisted recommendations has an overall tendency to decrease in most cases (although there are some

[3] In summary, AT and ST affect the number of recommendations being rewarded (because more agents are rewarded if their values are small). If an agent is frequently rewarded then it obtains more learning feedback and so can chase the equilibrium faster. A further effect of AT and ST is that the equilibrium price stability trades off with the speed of price convergence. This is because, with a high value of AT and ST, only a small amount of learning information is available for the agents (because fewer agents are rewarded). Hence, the agents need more adjustments to chase the equilibrium. The variables Y, Z and ΔP also affect an agent's speed of chasing the equilibrium. With high value of these variables, an agent alters its price rapidly to reach the equilibrium. Again, however, this trades off with the stability of its bids.

exceptions). Fig. 6(b) shows the average user-perceived quality of ten continuous auctions after the convergence (from 60^{th} to 69^{th}). By averaging over these auctions, we can see that the user-perceived quality decreases monotonically. Thus, Fig. 6 tells us the important information that our market mechanism is indeed capable of shortlisting the best recommendations in decreasing order of user-perceived quality.

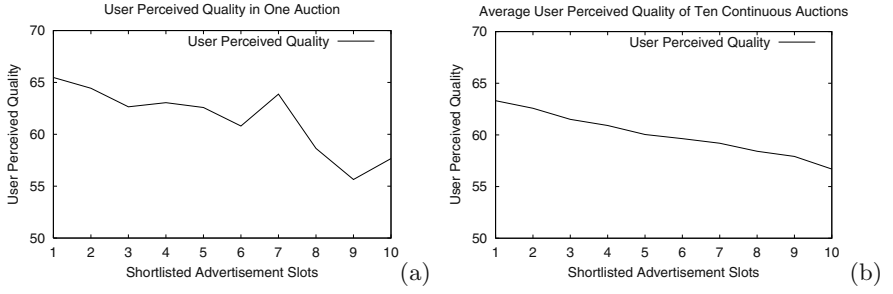

Fig. 6. User Perceived Quality of Shortlisted Recommendations.

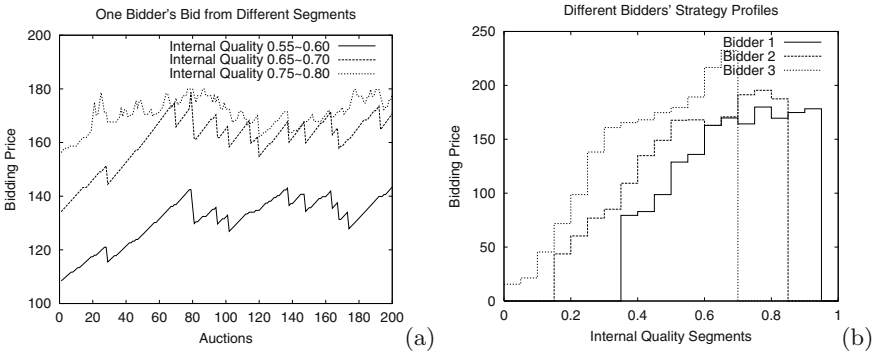

Fig. 7. Strategy Profile.

Clear Incentive. Now we aim to see if the recommending agents can relate their bids to the internal quality of their recommendations (meaning can an agent generate a steady strategy profile?). Again, we use the results from the price convergence experiments. In this case, each recommending agent builds up its strategy profile from its knowledge about the bids with respect to its 20 internal quality segments. Specifically, Fig. 7(a) shows the bidding prices for different segments of the recommending agent with the internal quality distribution $N(0.65, 0.1^2)$ in the above simulation. From Fig. 7(a), we can see that the price from the segment 0.65∼0.70 (with the majority of samples) keeps rising before the 60^{th} auction (the market convergence) and oscillates around 160 (the equilibrium price for recommendations from the segment 0.65∼0.70) afterwards. This is because the agent raises its price from a low initial bid to chase the equilibrium

price before the market converges and alters its price to minimize the difference between its bids and the equilibrium price so as to maximize its revenue. From Fig. 7(a), we can also see that the agent's prices for segments 0.55∼0.60 and 0.75∼0.80 converge at about 130 and 170 respectively. Fig. 7(a) indicates that the agent is capable of "learning" from the marketplace to alter its bids to certain levels in order to chase the equilibrium price. The solid line in Fig. 7(b) (marked "Bidder 1") plots this agent's strategy profile and the dashed and dotted lines represent the strategy profiles of the two recommending agents with the internal quality distributions $N(0.5, 0.1^2)$ and $N(0.35, 0.1^2)$. From our observation of various simulations, a recommending agent's strategy profile changes quickly before the market converges and then becomes relatively stable after the convergence. This tells us that the marketplace delivers a clear incentive to recommending agents enabling them to bid rationally and that agents with different recommendation methods are capable of relating their bids to their internal qualities by forming a steady strategy profile.

Now we are going to remove the second assumption made when designing the user agent in subsection 3.1. Once the first assumption holds (meaning there is a steady relationship $Q = \mathbf{f}(q)$ given that $q \sim N(\mu, \sigma^2)$) irrespect of whatever $\mathbf{f}$ is (whether $k > 0$ or not and whether $\mathbf{f}$ is linear or not), given a specific value of q, a steady value of Q ($= \mathbf{f}(q)$) always arises. Therefore, an agent with this specific method can always alter its price of q to the equilibrium price with respect to Q if there are sufficient samples from the internal quality segment containing q. For example, if the user agent assigns Q to recommendations from the agent in Fig. 7(a) following a behaviour of $Q = k \cdot q + b$ ($k < 0$), the solid line in Fig. 7(b) will have an overall tendency to decrease. Therefore, the recommending agents can always generate their strategy profiles without assumption 2 once assumption 1 holds.

Stability. To evaluate the stability of the market with respect to bidding strategies, we now consider what happens if some of the agents take a greedy strategy (bid as much as possible to outbid others). To this end, we select one recommending agent as the greedy bidder and the other agents still take the strategy introduced in subsection 2.2. All recommending agents are endowed with an initial credit of 65535. The greedy bidder always bids much higher than the others to get its recommendations shortlisted so as to make profit. However, this greedy bidder does not receive any more rewards from its recommendations when compared with the rewarded recommendations provided by the other non-greedy bidders. This is because the reward is not based on the bid price, but rather on the user-perceived quality. With the same amount of reward with respect to the same level of user-perceived quality, however, the greedy bidder pays much more for each of its shortlisted recommendations. Therefore, the greedy bidder goes bankrupt over time as shown in Fig. 8(b), while the other non-greedy bidders keep increasing their balance steadily. In comparison, when no greedy bidders participate, all recommending agents keep increasing their balance as shown in Fig. 8(a). Therefore, this experiment indicates that no greedy bidders can survive in our system.

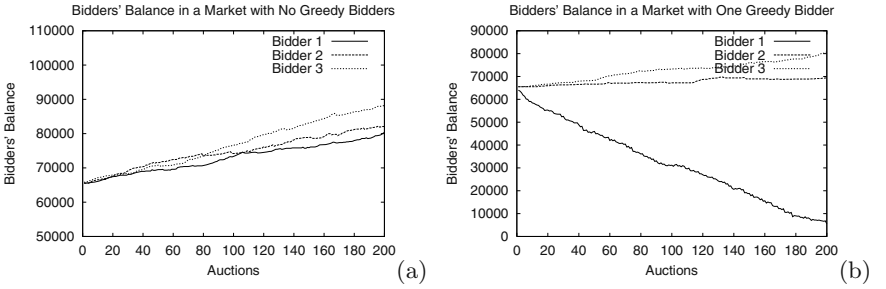

Fig. 8. Greedy versus Non-Greedy Bidder.

Fairness. We expect the market to be fair to all recommending agents irrespectively of the recommendation method they use. To this end, we organize nine recommending agents taking three different recommendation methods in the marketplace. From Fig. 9(a) it can be seen that the curves that represent the number being shortlisted (including both being rewarded and shortlisted but not rewarded) for each agent are close to each other. This, in turn, means all agents have an equal opportunity of being shortlisted. Thus the market is fair. Fig. 9(b) shows that Bidder 3 is rewarded fewer times than the others (although this is because its recommendations are of lower user-perceived quality). This, in turn, highlights the fact that a fair market does not necessarily guarantee equal opportunity of being rewarded. The opportunity of being rewarded depends on the user-perceived quality.

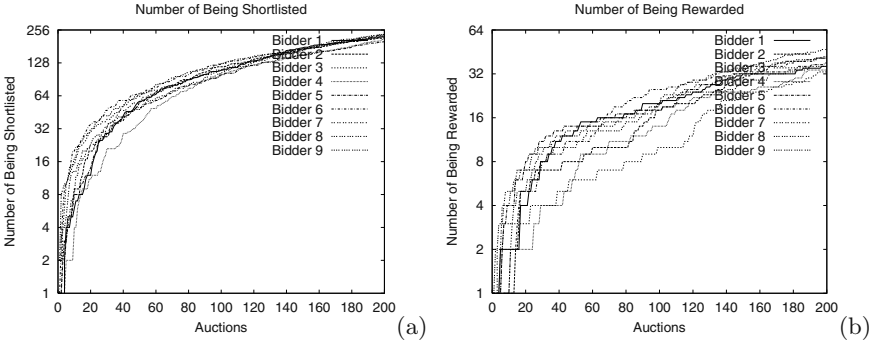

Fig. 9. Opportunity of Being Shortlisted and Rewarded.

3.3 System Properties

From the simulations demonstrated in subsection 3.2, we have shown that our marketplace can always reach an equilibrium when the shortlisted prices converge at different levels with respect to different user-perceived quality levels. After the price converges, the marketplace is capable of shortlisting recommendations in decreasing order of user-perceived quality. Also, after the convergence, all

recommending agents can build up their strategy profiles based on their bidding and the knowledge about the user's preference with respect to their internal quality. Having achieved this, all agents can relate their bids to the internal quality of their recommendations. In addition, the marketplace is stable and shows fairness to all recommending agents irrespectively of the bidding strategies and recommendation methods they utilize.

4 Related Work

A number of information filtering tools have been developed to cope with the problem of information overload. For example, SIFT [12] is a large scale information dissemination service capable of directing information of users' interests to the subscribed users. [13] demonstrates a learning approach to personalized information filtering using relevance feedback and genetic algorithms. However, these systems tend to filter based on document content and in many cases in our Web browsing domain issues such as quality, style and other machine unparsable properties are the key to giving good recommendations [14].

Thus, recommender systems [1] have been advocated. GroupLens [15] is a system using collaborative filtering [4] to make recommendations of Usenet news to help people find articles they will like in the huge stream of available articles. Letizia [16] was an early agent-based recommender system which is also coupled to a Web browser; it has the additional feature of scouting ahead of the user's current position on the Web. However, while such recommender systems tackle the weaknesses of content-based filtering techniques, each system employs a variety of techniques that are more or less successful for particular users in particular contexts. For this reason, we believe effective recommender systems should incorporate a wide variety of such techniques and that some form of overarching framework should be put in place to coordinate the various recommendations so that only the best of them (from whatever source) are presented to the user. Hence, our use of the marketplace is to perform this role.

The work most related to our own is that of [11] which uses a market to competitively allocate users' attention space (the user's attention to the shortlisted recommendations which is regarded as a scarce resource to be competed for by a number of recommenders). This work develops an adaptive bidding strategy that the agents can use to learn the user's preferences. However, this work and our own use the market mechanisms in different ways to solve the resource allocation problem in recommender systems. The marketplace in [11] is used only to coordinate agents bidding. Our market is used not only to this purpose but also to correlate the component recommenders' internal qualities to the user's perceived quality of recommendations.

5 Conclusions and Future Work

This paper has outlined the design and evaluation of a recommender system that uses market-based techniques to incorporate multiple recommendation methods

into a coherent framework. The market works as a means of coordinating various recommendation methods in an overarching system and ensures the peak performance of the overall system. Having differentiated the quality of recommendations from the point of view of the user and the individual recommendation method, the market works as a means of correlating these two qualities and supports the recommending agents' learning capabilities. Our various simulations have shown that the marketplace works efficiently in making good recommendations across multiple recommendation methods and the theoretical properties of the mechanism (as designed in [10]) do indeed hold. Specifically, the marketplace is capable of shortlisting the best recommendations in decreasing order of user-perceived quality. The marketplace is also capable of giving the recommending agents incentives to bid rationally in chasing the equilibrium price with respect to the user-perceived quality of their recommendations. This, in turn, means that the bidding agents can relate their bids to their internal quality. Moreover, the market is stable and is fair to all recommending agents. Having shown these results in simulation, the next and final step is to test the system's operation with real users in a practical application context.

References

1. Resnick, P., Varian, H.R.: Recommender Systems. Communications of the ACM **40** (1997) 56–58
2. Salton, G.: Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley, Reading, Massachusetts (1989) ISBN 0-201-12227-8.
3. El-Beltagy, S., Hall, W., DeRoure, D., Carr, L.: Linking in context. In: Proc. of the Twelfth ACM conference on Hypertext and Hypermedia, Denmark (2001) 151–160
4. Goldberg, D., Nichols, D., Oki, B., Terry, D.: Using collaborative filtering to weave an information tapestry. Communications of the ACM **35** (1992) 61–70
5. DeRoure, D., Hall, W., Reich, S., Pikrakis, A., Hill, G.J., Stairmand, M.: Memoir - an open framework for enhanced navigation of distributed information. Information Processing and Management **37** (2001) 53–74
6. Herlocker, J., Konstan, J., Terveen, L., Riedl, J.: Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems **22** (2004) 5–53
7. Breese, J., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: Proc of the Fourteenth Conference on Uncertainty in Artificial Intelligence, US (1998) 43–52
8. Moreau, L., Zaini, N., Zhou, J., Jennings, N.R., Wei, Y.Z., Hall, W., DeRoure, D., Gilchrist, I., O'Dell, M., Reich, S., Berka, T., Napoli, C.D.: A market-based recommender system. In: Proc. of the Fourth International Workshop on Agent-Oriented Information Systems (AOIS-2002), Italy (2002) 50–67
9. Clearwater, S.H., ed.: Market-Based Control. A Paradigm for Distributed Resource Allocation. World Scientific (1996)
10. Wei, Y.Z., Moreau, L., Jennings, N.R.: Recommender systems: A market-based design. In: Proc. of the Second International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS03), Australia (2003) 600–607
11. Bohte, S.M., Gerding, E., Poutré, H.L.: Competitive market-based allocation of consumer attention space. In: Proc. of the Third ACM conference on Electronic Commerce, US (2001) 202–205

12. Yan, T., Garcia-Molina, H.: SIFT—A tool for wide-area information dissemination. In: Proc. 1995 USENIX Technical Conference, US (1995) 177–186
13. Sheth, B., Maes, P.: Evolving agents for personalized information filtering. In: Proc. of the Nineth Conference on Artificial Intelligence for Applications (CAIA'93), US (1993) 345–352
14. Shardanand, U., Maes, P.: Social information filtering: algorithms for automating "word of mout". In: Proc. of the ACM Conference on Human Factors in Computing Systems (CHI'95). (1995) 210–217
15. Konstan, J.A., Miller, B.N., Maltz, D., Herlocker, J.L., Gordon, L.R., Riedl, J.: Grouplens: applying collaborative filtering to usenet news. Communications of the ACM **40** (1997) 77–87
16. Lieberman, H.: Letizia: An agent that assists web browsing. In: Proc. of the International Joint Conference on Artificial Intelligence, Canada (1995)

Comparing Agent-Oriented Methodologies

Khanh Hoa Dam and Michael Winikoff

RMIT University, Melbourne, Australia
{kdam,winikoff}@cs.rmit.edu.au
http://www.cs.rmit.edu.au/agents

Abstract. Numerous methodologies for developing agent-based systems have been proposed in the literature. However, their application is still limited due to their lack of maturity. Evaluating methodologies' strengths and weaknesses plays an important role in improving them and in developing the "next-generation" of methodologies. This paper presents a comparison of three prominent agent-oriented methodologies: **MaSE**, **Prometheus** and **Tropos**. It is performed based upon an attribute-based framework which addresses four major areas: *concepts*, *modelling language*, *process* and *pragmatics*. The objectivity of the comparison is increased by including inputs from the authors of the methodologies using a questionnaire and by conducting an experimental evaluation of the methodologies.

1 Introduction

> "*One of the most fundamental obstacles to large-scale take-up of agent technology is the lack of mature software development methodologies for agent-based systems.*" [15, page 11].

Even though many Agent Oriented Software Engineering (AOSE) methodologies have been proposed, few are mature or described in sufficient detail to be of real use. We believe that the area of agent-oriented methodologies is maturing rapidly and that the time has come to begin drawing together the work of various research groups with the aim of developing the "next generation" of agent-oriented software engineering methodologies.

A crucial step is to understand the relationship between the various key methodologies, and particularly to understand each methodology's strengths, weaknesses and domains of applicability. In this paper we perform the comparison on several well-known methodologies. These were selected since they (a) were described in more detail (e.g. journal paper rather than a conference paper); and (b) were perceived as being significant by the agent community. Another important factor was whether the methodology had been developed over an extended time period based on feedback from users other than the developers of the methodology. Based on these criteria we chose the five[1] methodologies Gaia [29, 30], MESSAGE [3, 4], MaSE [8, 9], Prometheus [17–20, 27] and Tropos [1, 12]. However due to space limitations only **MaSE**, **Prometheus** and **Tropos** are presented in this paper[2].

[1] We were limited to choosing five methodologies since we had that many summer studentships. This prevented us from being able to compare all of the methodologies that met the selection criteria.

[2] The evaluation and comparison of all five methodologies can be found in [7].

P. Giorgini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 78–93, 2004.

In section 2, we briefly introduce these methodologies. Since it is impossible to accurately summarise a detailed methodology in a single page we attempt to give a flavour of each methodology and outline the process and notations used. We refer the reader to original sources for further details on each methodology.

We then (section 3) describe a framework for comparing AOSE methodologies and, in section 4, apply the framework to compare the methodologies.

In doing this there are two key issues. Firstly, how can we ensure that the framework is unbiased and complete (covers all significant criteria); and, secondly, how can we avoid the comparison being affected by our biases? In particular, one of the authors of this paper is also one of the developers of the Prometheus methodology.

The first issue is addressed by basing the framework on existing work in the comparison of Object-Oriented (OO) methodologies. By including a range of issues that have been identified as important by a range of authors we avoid biasing the comparison by including only issues that we consider important. The second issue we address by having others do the assessment of each methodology. Specifically, for each methodology we asked the authors of the methodology to fill in a questionnaire assessing the methodology. We also had students each develop designs for the same application using different methodologies. We collected comments from the students as they developed their application designs over summer (Dec. 2002 – Feb. 2003) as well as asking them to fill in the questionnaire. The aim of the questionnaire was *not* to provide a statistically significant sample in order to carry out a scientific experiment. This was not practical. Rather, the aim was to avoid any particular bias by having a range of viewpoints.

The experimental application was a Personal Itinerary Planner System (PIPS). Its main goal is to assist a traveller in finding activities. Suppose you are visiting an unfamiliar city and have nothing planned on a weekend or evening. Rather than spend a day wondering aimlessly (or worse, working in a hotel room), you access PIPS. After telling PIPS where you are, when you are looking for things to do and what are your interests, it responds with a number of itineraries. An itinerary consists of one or more activities such as eating at a restaurant, going to a show, visiting a local attraction (e.g. zoo, aquarium, historic site, etc.). However, it is not just a collection of activities. The itinerary should meet constraints of time and space. Where an activity is followed by another activity at a different location, they should be separated by a transport activity that takes the user from the location of the first activity to the location of the second.

2 The Methodologies

2.1 MaSE

The goal of *Multiagent Systems Engineering* (MaSE) [9] is to provide a complete-lifecycle methodology to assist system developers to design and develop a multi-agent system. It fully describes the process that guides a system developer from an initial system specification to system implementation. This process consists of seven steps, divided into two phases.

The MaSE **Analysis** stage includes three smaller process steps. First, the *Capturing Goals* step guides the analysts to identify goals and structure and represent them as a *goal hierarchy*. The second step, *Applying Use Cases*, involves extracting main scenarios from

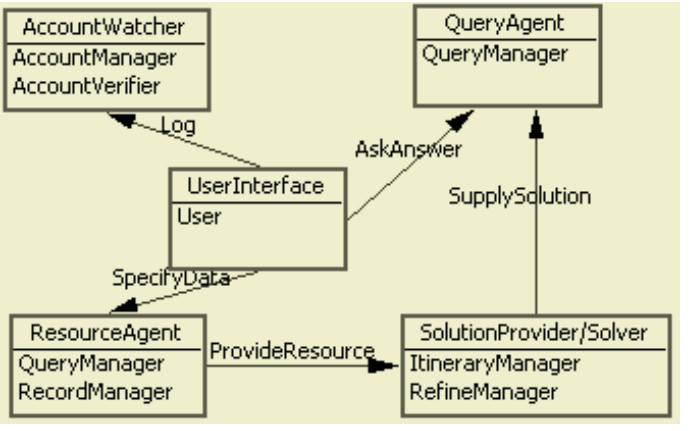

Fig. 1. PIPS agent class diagram (extracted from the PIPS design documentation produced by Yenty Frily using agentTool).

the initial system context or copying them from it if they exist. These use cases are also used to build a set of *sequence diagrams* (similar to UML sequence diagrams). *Refining Roles* is the final step of the Analysis phase where a *Role Model* and a *Concurrent Task Model* are constructed. The Role Model describes the roles in the system. It also depicts the goals for which those roles are responsible, the tasks that each role performs to achieve its goals and the communication path between the roles. Tasks are then graphically represented in fine-grained detail as a series of finite machine automata in the Concurrent Task Model.

The first step of the **Design Phase** is called "*Creating Agent Classes*". The output of this step is an *Agent Class Diagram* that describes the entire multi-agent system. The agent class diagram shows agents and the roles they play. Links between agents show conversations and are labelled with the conversation name. For example, in Figure 1 the agent AccountWatcher comprises two roles (AccountManager and AccountVerifier) and it converses with the UserInterface agent type. The details of the conversations are described in the second step of the design phase ("*Constructing Conversations*") using *communication class diagrams*. These are a form of finite state machine. The third step of the Design stage is *Assembling Agent Classes*. During this step, we need to define the agent architecture and the components that build up the architecture. In terms of agent architecture, MaSE does not dictate any particular implementation platform. The fourth and final step of the design phase is *System Design*. It involves building a *Deployment Diagram* which specifies the locations of agents within a system.

MaSE has extensive tool support in the form of agentTool [8]. Its latest version 2.0 [3] implements all seven steps of MaSE. It also provides automated support for transforming analysis models into design constructs.

[3] `http://www.cis.ksu.edu/~sdeloach/ai/agentool.htm`

2.2 Prometheus

The *Prometheus* [17–20, 27] methodology is a detailed AOSE methodology that is aimed at non-experts. It has been successfully taught to and used by undergraduate students. Prometheus consists of three phases: *system specification*, *architectural design* and *detailed design*.

The first phase of Prometheus, the **system specification** phase, involves two activities: *determining the system's environment* and *determining the goals and functionality of the system*. The system's environment is defined in terms of *percepts* (incoming information from the environment) and *actions* (the means by which an agent affects its environment). In addition, external data are defined. Defining the system's functionality is done by identifying goals, by identifying functionalities that achieve these goals, and by defining use case scenarios. Use case scenarios describe examples of the system in operation. A typical scenario includes a sequence of steps depicting incoming percepts, messages sent and the activities and actions. The development of goals, functionalities and use case scenarios is usually iterative. Each of the concepts is captured using a descriptor form.

The second stage, **architectural design**, involves three activities: *defining agent types*, *designing the overall system structure*, and *defining the interactions between agents*. Agent types are derived by grouping functionalities. The choice of grouping is guided by consideration of coupling and cohesion which are identified with the aid of the data coupling diagram and agent acquaintance diagram. Each identified agent type is described using an agent descriptor which describes the lifecycle of this agent type (how and when it is initialized and destroyed), its functionalities, the data it uses and produces, its goals, the events it should respond to, its actions and the other agent types that it interacts with. The system's structure is captured in a *system overview diagram*, arguably the single most important design artefact in Prometheus. The system overview diagram provides the designers and implementers a general picture of how the system as a whole will function. It shows the agent types, the communication links between them and data. It also shows the system's boundary and its environment (actions, percepts and external data). An example of PIPS's system overview diagram is shown in Figure 2. Whereas the system overview diagram captures the static structure of the system, interaction protocols capture the dynamic behaviour of the system by defining the intended valid sequences of messages between agents. The interaction protocols are developed from interaction diagrams, which, in turn, are based on the scenarios.

The internals of each agent and how it will accomplish its tasks within the overall system are addressed in the **detailed design** phase. It focuses on *defining capabilities*, *internal events*, *plans* and *detailed data structure* for each agent type identified in the previous step. Firstly, an agent's capabilities are depicted via a capability descriptor which contains information such as which events are generated and which events are received. The capability descriptor also includes a description of the capability, details involving interactions with other capabilities and references to data read and written by the capability. Secondly, at a lower level of detail, there are other types of descriptors: individual plan descriptors, event descriptors and data descriptors. These descriptors provide the details so that they can be used in the implementation phase. The detailed design phase also involves constructing agent overview diagrams. These are very similar

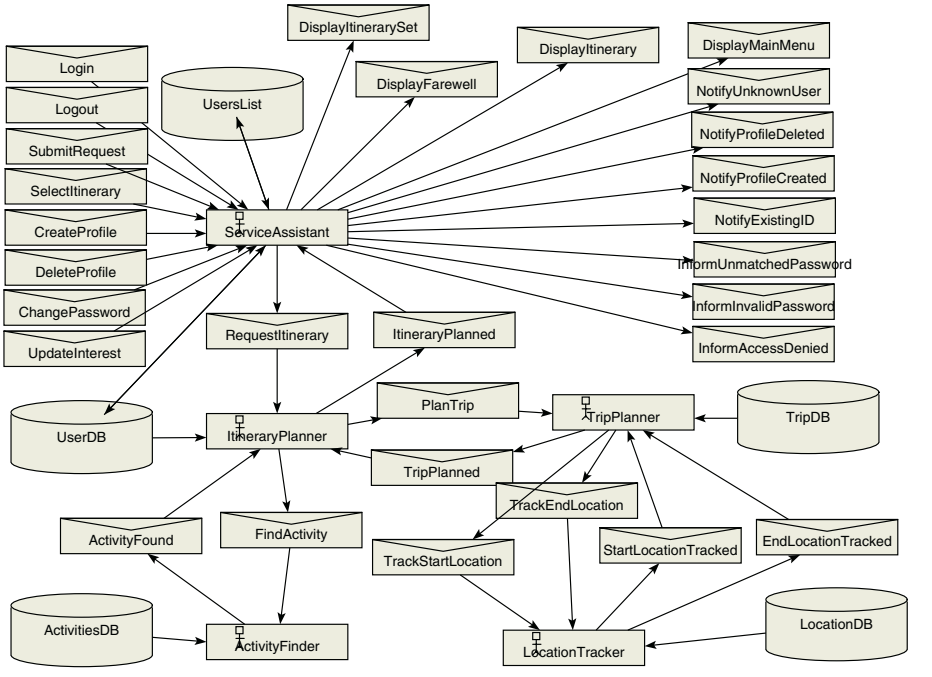

Fig. 2. PIPS System Overview (extracted from the PIPS design documentation produced by Robert Tanaman using the Prometheus Design Tool (PDT)).

to the system overview diagram in terms of style but give the top level view of each agent's internals rather than the system as a whole. Agent overview diagrams, together with the capability descriptors, provides a high level view of the components within the agent internal architecture as well as the their connectors (interactions). They show the top level capabilities of the agent, the flow of tasks between these capabilities and data internal to the agent.

Prometheus is supported by two tools. The JACK Development Environment (JDE), developed by Agent Oriented Software (*www.agent-software.com*) includes a design tool that allows overview diagrams to be drawn. These are linked with the underlying model so that changes made to diagrams, for example adding a link from a plan to an event, are reflected in the model and in the corresponding JACK code. The Prometheus Design Tool (PDT) provides forms to enter design entities. It performs cross checking to help ensure consistency and generates a design document along with overview diagrams. Neither PDT nor the JDE currently support the system specification phase.

2.3 Tropos

Tropos [1, 12] is an agent-oriented software development methodology created by a group of authors from various universities in Canada and Italy. One of the significant differences between Tropos and the other methodologies is its strong focus on early

requirements analysis where the domain stake-holders and their intentions are identified and analysed. This analysis process allows the reason for developing the software to be captured. The software development process of Tropos consists of five phases: *Early Requirements*, *Late Requirements*, *Architectural Design*, *Detailed Design* and *Implementation*.

Early Requirements: The requirements phase of Tropos is influenced by Eric Yu's i* modelling framework [31]. Tropos uses the concept of actor and goals to model the stake-holders in the target domain and their intentions respectively. Tropos divides goals into two different types. Hardgoals eventually lead to functional requirements whilst softgoals[4] relate to non-functional requirements. There are two models that represent goals and actors at this point in the methodology. First, the actor diagram depicts the stake-holders and their relationship in the domain. The latter are called "social dependencies" that reflect how actors depend on one another for goals to be accomplished, plans to be executed and resources to be supplied. Second, the goal diagram shows the analysis of goals and plans with regard to a specific actor who has the responsibility of achieving them. Goals and plans are analysed based upon several reasoning techniques as proposed by the methodology such as: means-end analysis, AND/OR decomposition and contribution analysis. These techniques help the analysts structure the system's goals, identify softgoals, plans and resources providing the means for accomplishing a goal and capture goals that promote or interfere with the fulfilment of other goals.

Late Requirements: This phase involves extending the models that were created in the previous step. The importance of this stage is the modelling of the target system within its environment. The system-to-be is modelled as one or more actors. Its interdependencies with other actors in the models contribute to the accomplishment of stake-holder goals. Therefore, these dependencies define the target system's functional and non-functional requirements. Figure 3 show the dependency of Tourism Commission on PIPS to provide information (hard goal). It also requires a usable PIPS (softgoal). These goals are then decomposed into subgoals. For instance, the goal "provide information" is fulfilled by the composite achievement of two sub-goals "search information" and "generate output". The subgoal "search information", in turn, has several sub-goals such as "access to tourism database", and "access to user's records". Additionally, the positive contribution of other goals to the softgoal "easy to use" is also shown. A "user-friendly interface" that offers "simplicity" and provides guidelines promotes the fulfilment of the goal "easy to use".

Architectural Design: Tropos defines three steps that system designers can apply to proceed through this phase. At the first step, new actors are included and described by an extended actor diagram. These new actors are derived based on the choice of architectural style. They may exist to fulfil non-functional requirements or to support sub-goals decomposed in the previous steps. The second and third steps respectively identify the capabilities and group them to form agent types, where each agent types is formed by joining some number of capabilities.

Detailed Design: The Tropos detailed design phase involves defining the specification of agents at the micro level. There are three different types of diagrams that the designers

[4] Softgoals are goals whose satisfaction conditions cannot be precisely defined.

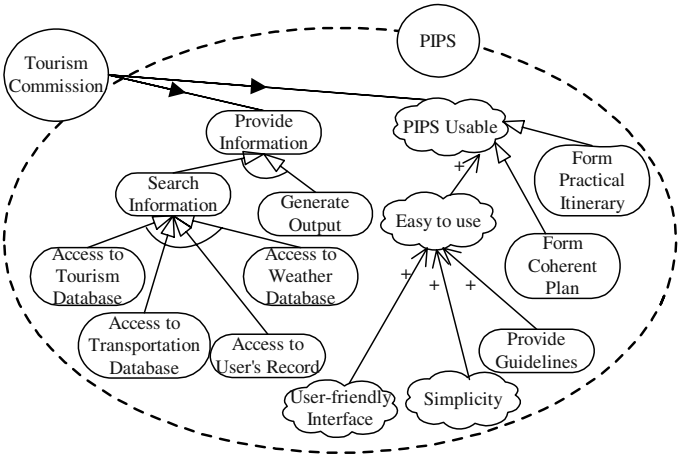

Fig. 3. Goal Diagram – PIPS in connection with Tourism Commission (re-drawn based on the PIPS design documentation produced by Sindawati Hoetomo).

need to produce to depict the capabilities, the plans of agents and the interactions between them. Tropos uses UML activity diagrams to represent capabilities and plans at the detailed level. Plan diagrams are fine-grained representations of each plan node in the capability diagrams. The interaction between agents in the system is represented by agent interaction diagrams. They are in fact AUML interaction diagrams.

Implementation: Having finished the detailed design stage, we can now move to the final step of Tropos, the Implementation phase. Tropos chooses a BDI platform, specifically JACK Intelligent Agents$^{(TM)}$ [2], for the implementation of agents. JACK provides five main language constructs: agents, capabilities, database relations, events and plans. At this stage, developers need to map each concepts in the design phase to the five constructs in JACK. Tropos provides several guidelines and heuristics for mapping Tropos concepts to BDI concepts and BDI concepts to JACK constructs.

3 A Comparison Framework

In this section, we briefly describe a methodology evaluation framework within which the methodology comparison is conducted. The framework consists of a set of criteria which addresses not only classical software engineering attributes but also properties that are uniquely found in AOSE. In order to avoid using an inappropriate comparison framework, the properties in our framework are derived from a survey of work on comparing AOSE methodologies [5, 16, 24–26], and more importantly on comparing OO methodologies [10, 11, 13, 22, 23, 28]. The comparison framework covers four major aspects of each AOSE methodology: **Concepts**, **Modelling language**, **Process** and **Pragmatics**. This framework is adapted from a framework proposed in [10] for comparing Object-Oriented Methodologies. In addition to the four above components, that

framework considers two more areas: Support for Software Engineering and Marketability. For our framework, we have decided to address "Support for Software Engineering" criteria in various places in the above four major aspects. With regard to "Marketability" issues, since all of our compared AOSE methodologies are still being researched and developed, we do not believe that marketability criteria are applicable.

Concepts: Agent-oriented concepts are of great importance for agent-oriented methodologies in general and for agent-oriented modelling languages in particular. Based on the literature research, we have found a set of significant agent-oriented concepts that are commonly addressed. These include the definition of agents, their characteristics such as autonomy, adaptability, mental notions (such as beliefs, desires and intention), the relationship and communication between agents and other concepts such as goals, agent roles and capabilities, as well as percepts, actions and events. When reviewing each AOSE methodology's definition of those agent-oriented concepts, we essentially look at the extent to which the methodology supports the concept or to which it supports the construction of agents that possess the attribute.

Modelling Language: If agent-oriented concepts are the basis for any AOSE methodology, then the modelling language for representing designs in terms of those concepts is generally the core component of any software engineering methodology. A typical modelling language consists of three main components [11]: symbols (either graphical or textual representation of the concepts), syntax and semantics. It is important that the modelling language allows the system under development to be modelled from different views such as behavioural, functional and structural views [28]. As a result, by having a good modelling notation, the methodology effectively eases the complex tasks of requirement analysis, specification analysis and design. Therefore, measuring the quality of the modelling language of an AOSE methodology plays an important part in our evaluation.

The criteria which assess the modelling language of each methodology are categorised into two groups. *Usability* criteria reflect usage requirements of a modelling language in terms of providing a means for software developers to exchange their thoughts and ideas. These criteria basically address the question of how easy the notation and the models are to understand and to use [22, 23]. They address the complexity, clarity and understandability of a modelling language. In addition we also assess whether the modelling notation is adequate for expressing all the necessary concepts and whether it is expressive, that is whether these concepts are expressed in a natural and direct way.

The second group of criteria to assess a modelling language is *technical criteria*. They involve the **unambiguity** and **consistency** of a modelling language. **Unambiguity** means that a constructed model can be interpreted unambiguously whereas **consistency** is a technical quality relating to the assistance of a modelling technique to the software designer in guaranteeing that between representations, no set of individual requirements is in conflict [28]. The technical qualities of a modelling language also concern its ability to support **traceability**. This is the ability to track dependencies between different models and between models and code.

Process: As discussed above, the modelling language is considered as a mandatory part of any software engineering methodology. However, in constructing a software system,

software engineering also emphasizes the series of activities and steps performed as part of the software life cycle [10, 13, 28]. These activities and steps form the process that assists system analysts, developers and managers in developing software. According to [10], an ideal methodology should cover six stages, which are enterprise modelling, domain analysis, requirements analysis, design, implementation and testing. Methodologies that cover all aspects of the system development are more likely to be chosen because of the consistency and completeness they provide.

An important aspect in evaluating whether a methodology covers a particular process step is the degree of detail provided. It is one thing to mention a step ("at this point the designer should do X") and another thing to provide a detailed description of *how* to perform X. Since design is inherently about tradeoffs, detailed descriptions are usually expressed using heuristics rather than algorithms, as well as examples. Thus, in assessing support for process steps, we identify whether the step is mentioned, whether a process for performing the step is given, whether examples are provided and whether heuristics are given. In addition, it is necessary to consider which development contexts are supported by the methodology. In particular, we need to examine whether a particular AOSE methodology supports legacy system integration. This criterion is important, especially for AOSE because, as emphasized in [14], one of the key pragmatic issues which determine whether the agent-oriented paradigm can be popular as a software engineering paradigm is the degree to which existing software can be integrated with agents. In addition to heuristics, the availability of estimating guidelines is essential in aiding the project planning tasks. Hence, we also investigate whether the assessed methodology provides estimating guidelines such as: the estimating the costs, schedule, etc. of the developed system.

Pragmatics: In addition to issues relating to notation and process, the choice of an AOSE methodology depends on the *pragmatics* of the methodology. This can be assessed based on two aspects [16]: management and technical issues. Management criteria should consider the support that a methodology provides to management when adopting it. They include the cost involved in selecting the new methodology, its maturity and its effects on the current organization business practices [10, 16, 28]. There are different types of cost associated with adopting the methodology such as the cost of acquiring methodology and tool support, as well as the required training to fully exploit the methodology. The methodology maturity, on the other hand, concerns the resources available to support the methodology (e.g. documentation, training, consulting services) and the availability of automated tools. In addition, the history of the methodology's use is considered so that a methodology that has been used to create industrial strength applications should be preferred over ones that have only been used to develop small demonstration projects.

Differing from management issues, technical criteria look at a methodology from another angle. They consider whether the methodology is targeted at a specific type of software domain such as information systems, real time systems or component-based systems [16]. With regard to this issue, the methodology that is applicable to a wide range of software domains tends to be more preferred. Additionally, technical evaluation considerations measure the methodology's support for designing systems that are scalable. It means that the system should allow the incorporation of additional resources and software components with minimal user disruption.

4 Comparing Methodologies

Based on the above comparison framework, we have developed a questionnaire[5] consisting of around 60 questions addressing the concepts, modelling language, process and pragmatics of a methodology. The questionnaire was distributed to the authors of each AOSE methodology that we have selected to compare (MESSAGE, Gaia, MaSE, Tropos, and Prometheus). 12 responses (out of 16) have been received.

In addition to obtaining evaluations from the authors of the methodologies, we also had a number of students who, over the summer, designed an agent application, each using a different methodology. Each student gave us feedback on their experience in understanding and using the methodology and also completed the questionnaire at the end of their work. For each of the three methodologies both the creators of the methodology responded to the questionnaire as did the users (summer students). The results are summarised in Figures 4 and 5 and discussed below.

Concepts & Properties	**MaSE**	**Prometheus**	**Tropos**
Autonomy	H/M/DK	H/NA/H	H/M/M
Mental attitudes	L/M/H	H/M/H	H
Proactive	H/M/H	H/M/DK	H
Reactive	M	H/M/DK	H/L/DK
Concurrency	H/M/H	M/L/DK	H/M/H
Teamwork	H/M/H	N/L/NA	H/H/M
Protocols	H	M/H/M	NA/M/M
Situated	M/L/H	H	H
Clear concepts	SA/A/A	A/A/DA	SA/A/N
Concepts overloaded	A/N/SA	N	SDA/N/DA
Agent-oriented	SA/A/A	SA	SA/A/SA

Fig. 4. Comparing a methodology's properties, attributes, process and pragmatics. Notation: L for Low, M for medium, H for High, DK for Don't Know, SDA for Strongly Disagree, DA for Disagree, NA for Not Applicable, N for Neutral, A for Agree, SA for Strongly Agree, _ for no response. The first two entries in each column are the developers of the methodology, the third is the student. A single entry in the column indicates that all three answers agreed.

Concepts: With regard to agent-oriented concepts, the level of support for autonomy of all of the methodologies is overall good (ranging from medium to high). Prometheus & Tropos support very well the use of mental attitudes (such as beliefs, desires, intentions) in modelling agents' internals (medium to high) whereas MaSE provides weaker support. The questionnaire also addresses the support for pro-activeness and reactiveness, although it seems that these two attributes are difficult to measure even though they seem to be fairly well supported by all three methodologies (medium-high for MaSE and

[5] The questionnaire can be found at
`http://yallara.cs.rmit.edu.au/~kdam/Questionnaire/Questionnaire.html`

Concepts & Properties	MaSE	Prometheus	Tropos
Modelling & Notation			
Static+Dynamic	SA/A/A	SA/A/A	N/A/A
Syntax defined	A/A/SA	SA/A/A	SA/N/A
Semantics defined	A/SA/SA	A	SA/A/A
Clear notation	A	SA/A/A	SA/A/N
Easy to use	SA/A/A	A/N/A	SA/A/N
Easy to learn	N/N/A	SA/NA/SA	SA/N/A
Different views	N/N/A	A/A/SA	SA/A/N
Language adequate & expressive	SA/N/N	A	SA/A/N
Traceability	A/SA/SA	A	A/N/A
Consistency check	SA/A/SA	SA/A/A	_/A/DA
Refinement	SA/A/A	SA	SA/A/DA
Modularity	SA/A/A	SA/SA/A	SA/A/N
Reuse	N/SA/A	N/A/N	_/A/DA
Hierarchical modelling	N/A/A	SA/A/A	SA/A/DA
Process			
Requirements	SPEH	SPEH	SPE
Architectural design	SPEH	SPEH	SPE
Detailed design	SPEH	SPEH	SPE
Implementation	SEH/SPE/S	SPEH/S/n	SE/SPE/SPEH
Testing & Debugging	SPE/n/n	SPEH/S/n	n
Deployment	SE/SPE/SPEH	n	n
Maintenance	n/SPE/n	n	n
Pragmatics			
Quality	N/DA/A	A/N/N	DA/A/_
Cost estimation	_/DA/SA	DA/DA/N	DA/N/_
Management decision	_/DA/SA	SDA/N/_	SA/A/_
apps	21+	6-20	1-5
Real apps	no	no	no
Used by non-creators	yes	yes	yes/no/no
Domain specific	no	no	yes/no/no
Scalable	_/N/N	N/A/N	N/N/_
Distributed	_/SA/SA	SA/A/N	N/A/_

Fig. 5. Comparing methodology's properties, attributes, process and pragmatics. Notation: L for Low, M for medium, H for High, DK for Don't Know, SDA for Strongly Disagree, DA for Disagree, NA for Not Applicable, N for Neutral, A for Agree, SA for Strongly Agree, _ for no response. S for Stage mentioned, P for Process given, E for Examples given, H for Heuristics given, n for none. The first two entries in each column are the developers of the methodology, the third is the student. A single entry in the column indicates that all three answers agreed.

Prometheus, mostly high for Tropos). In terms of support for concurrency, although the ratings are mostly medium-high and varied considerably, MaSE is probably best with its protocol analyser, and Prometheus was rated as being one of the weakest[6]. Although the methodologies all support cooperating agents, none of them support teams of agents in the specific sense of [6]. Both MaSE and Prometheus model the dynamic aspects of the system and handle protocols well. Tropos does not provide strong support for protocols or for modelling the dynamics of the system except for some support at the detailed design level. According to the questionnaire, the concepts used in the methodologies tend to be clearly explained and understandable. However, the student who used Prometheus responded that there was some confusing terminology such as the distinction between percept, incident, trigger and event. All three methodologies were perceived as being clearly agent-oriented.

Modelling Language: Overall, the responders felt that the methodologies' notations were clear and reasonably well defined (syntax/semantics) and fairly easy to use. Tropos was an interesting case: there was disagreement on whether the concepts were clear and whether the notation was clear and easy to use; furthermore, there was disagreement on whether the syntax was defined but, oddly, there was consensus that the semantics were defined. Although one respondent felt strongly that MaSE's notation was adequate and expressive, the other two respondents disagreed (neutral). MaSE also does not claim to support different views (neutral from both creators). To some extent, the modelling language of all the methodologies supports traceability, i.e. the ability to track dependencies between different models. In terms of consistency checking, the level of support differs between methodologies. MaSE and Prometheus support it well whereas Tropos does not appear to support it. Refinement, modularity and hierarchical modelling are generally well-supported (although there was disagreement from the student using Tropos), although reuse is not well handled by any of the methodologies.

Overall, the students had a very good impression of the notation of all the methodologies. For instance, Prometheus was highly appreciated and the system overview diagram in particular was found to be useful. The students also reported some minor issues. For example, the Capability Diagram in Tropos is hard to draw since it is presented differently in [1] and [12]. There are some cases where the amount of text on arcs in the MaSE's concurrent diagrams makes them hard to read.

Process: From the software development life-cycle point of view, all of the methodologies cover the requirements, architectural design and detailed design. The students' responses to these phases of the three methodologies are also positive. They all said that the analysis stage of the methodology they had used was well described and provided useful examples with heuristics. This helped them to shift from object-oriented thinking to agent-oriented. The implementation phase is, surprisingly, not well supported; for example Tropos briefly explains that the concepts developed during design map to JACK constructs but does not provide a detailed process, heuristics, examples or a discussion of the issues. Only MaSE and Prometheus mention testing/debugging. It is unclear to what extent MaSE supports it, while Prometheus' support is part of a research project [21] not

[6] Although we should note that the handling of protocols in Prometheus has been developed since the time of the questionnaire.

yet integrated into tools for use by developers. Only MaSE discusses deployment and the level of support is unclear. Only one respondent (for MaSE) indicated any support for maintenance.

Pragmatics: As we have mentioned earlier, the pragmatics of a methodology plays a very important role in determining its applicability in industry as well as in academia. In the questionnaire we asked the authors who the intended audiences for the methodology are. MaSE and Prometheus target undergraduate and industry programmers, whereas Tropos is aimed at experts. Furthermore, to measure how complex a methodology is to users, we used UML (Unified Modelling Language) and RUP (Rational Unified Process) as a benchmark. However, it is not clear that there is a consensus on the perceived complexity of UML+RUP, and so the answers to this question didn't allow any strong conclusions to be drawn. Regarding the availability of resources supporting the methodologies, most of them are in the form of conference papers and journal papers or tutorial notes. None of the methodologies are published as text books. None of the methodologies seem to address issues such as quality assurance or cost estimating guidelines. Although one respondent indicated that Tropos provides some support for decision making by management, e.g. when to move between phases, we do not agree with this assessment.

The availability of tool support also varies. MaSE and Prometheus are well supported with agentTool (MaSE) and JDE and PDT (Prometheus). Despite some minor issues, the use of agentTool really helped the student in drawing diagrams, checking model consistency and especially semi-automatically transforming analysis models to design constructs. PDT was also described by the student using it as being quite useful. Tropos has only weak tool support (a diagram editor). Although we attempted to determine how much "real" use (as opposed to student projects, demonstrators etc.) had been made of each methodology, it was not clear from the responses to what extent each methodology had been used, who had used the methodology and what it had been used for.

5 Conclusion

We have presented a comparison of three prominent methodologies. Overall, all three methodologies provide a reasonable support for basic agent-oriented concepts such as autonomy, mental attitudes, pro-activeness, reactiveness etc. They all are also regarded by their developers and the students as clearly agent-oriented. In addition, the notation of the three methodologies is generally good. Regarding the process, all the methodologies provide examples and heuristics to assist developers from requirements gathering to detailed design. Implementation was supported to some degree by all methodologies whereas testing/debugging and maintenance are not clearly well-supported by any methodology. Additionally, some important software engineering issues such as quality assurance, estimating guidelines and supporting management decisions are not supported by any of the methodologies.

5.1 Related Work

There has not been much work in comparing agent-oriented methodologies. Onn Shehory and Arnon Sturm [24] performed a feature-based evaluation of several AOSE methodologies. Their criteria included software engineering related criteria and criteria relating

to agent concepts. In another paper [25], they used the same techniques in addition to a small experimental evaluation to perform an evaluation of their own Agent Oriented Modelling Techniques (AOMT). This work suffers from subjectivity in that the criteria they identified are those that they see as important and, naturally, AOMT focuses on addressing these criteria.

A framework to carry out an evaluation of agent-oriented analysis and design modelling methods has been proposed by Cernuzzi and Rossi [5]. The proposal makes use of feature-based evaluation techniques but metrics and quantitative evaluations are also introduced. The significance of the framework is the construction of an attribute tree, where each node of the tree represents a software engineering criterion or a characteristic of agent-based system. Each attribute is assigned with a score and the score of attributes on the node is calculated based on those of their children. They have applied that framework to evaluate and compare two AOSE methodologies: the Agent Modelling Techniques for Systems of BDI (Belief, Desire and Intention) Agents and MAS-CommonKADS.

In [16], O'Malley and DeLoach propose a number of criteria for evaluating methodologies with a view to allowing organisations to decide whether to adopt AOSE methodologies or use existing OO methodologies. Although they performed a survey to validate their criteria, they do not provide detailed guidelines or a method for assessing methodologies against their criteria. Their example comparison (between MaSE and Booch) gives ratings against the criteria without justifying them. Their work is useful in that it provides a systematic method of taking a set of criteria, weightings for these criteria (determined on a case by case basis) and an assessment of a number of methodologies and determining an overall ranking and an indication of which criteria are critical to the result.

5.2 Further Work

So far, the three methodologies have been compared based on the set of properties or attributes regarding concepts, notation, process and pragmatics. In addition to this, we intend to perform a *structural* comparison of the methodologies. This will involve examining their processes and models in detail and looking at commonalities and differences. For example, capturing goals and use cases is a feature of the requirements phase of a number of methodologies and this suggests that they are a useful activity. These and other issues, if evaluated in detail, may contribute another step towards developing the "next generation" of agent-oriented methodologies.

Acknowledgements

We would like to thank Hiten Bharatbhai Ravani, Robert Tanaman, Sindawati Hoetomo, Sheilina Geerdharry and Yenty Frily for their work over the summer with the different methodologies. We would like to thank Lin Padgham for comments on a draft of this paper. We would also like to acknowledge the support of Agent Oriented Software Pty. Ltd. and of the Australian Research Council (ARC) under grant CO0106934.

References

1. Paolo Bresciani, Paolo Giorgini, Fausto Giunchiglia, John Mylopoulos, and Anna Perini. Troops: An agent-oriented software development methodology. Technical Report DIT-02-0015, University of Trento, Department of Information and Communication Technology, 2002.
2. Paolo Busetta, Ralph Rönnquist, Andrew Hodgson, and Andrew Lucas. JACK Intelligent Agents - Components for Intelligent Agents in Java. Technical report, Agent Oriented Software Pty. Ltd, Melbourne, Australia, 1998.
3. G. Caire, F. Leal, P. Chainho, R. Evans, F.G. Jorge, G. Juan Pavon, P. Kearney, J. Stark, and P Massonet. Project p907, deliverable 3: Methodology for agent-oriented software neginnering. Technical Information Final version, European Institute for Research and Strategic Studies in Telecommunications (EURESCOM), 09 2001.
4. Giovanni Caire, Francisco Leal, Paulo Chainho, Richard Evans, Francisco Garijo, Jorge Gomez, Juan Pavon, Paul Kearney, Jamie Stark, and Philippe Massonet. Agent oriented analysis using MESSAGE/UML. In Michael Wooldridge, Paolo Ciancarini, and Gerhard Weiss, editors, *Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001)*, pages 101–108, 2001.
5. L. Cernuzzi and G. Rossi. On the evaluation of agent oriented modeling methods. In *Proceedings of Agent Oriented Methodology Workshop*, Seattle, November 2002.
6. P. R. Cohen and H. J. Levesque. Teamwork. *Nous*, 25(4):487–512, 1991.
7. Khanh Hoa Dam. Evaluating and comparing agent-oriented software engineering methodologies. Masters minor thesis, School of Computer Science and Information Technology, RMIT University, Melbourne, Australia, June 2003. (supervisors: Michael Winikoff and Lin Padgham).
8. Scott A. DeLoach. Analysis and design using MaSE and agentTool. In *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001)*, 2001.
9. Scott A. DeLoach, Mark F. Wood, and Clint H. Sparkman. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.
10. Berard E.V. A comparison of object-oriented methodologies. Technical report, Object Agency Inc., 1995.
11. U. Frank. Evaluating modelling languages: relevant issues, epistemological challenges and a preliminary research framework. Technical Report 15, Arbetsberichte des Instituts fuer Wirtshaftsinformatik (Universität Koblenz-Landau), 1998.
12. Fausto Giunchiglia, John Mylopoulos, and Anna Perini. The Tropos software development methodology: Processes, Models and Diagrams. In *Third International Workshop on Agent-Oriented Software Engineering*, July 2002.
13. S. Hong, G. Van den Goor, and S. Brinkkemper. A formal approach to the comparison of object-oriented analysis and design methodologies. In *The Twenty-Sixth Annual Hawaii International Conference on System Sciences*, pages 689–699, Hawaii, 1993.
14. N. R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.
15. Michael Luck, Peter McBurney, and Chris Preist. Agent technology: Enabling next generation computing: A roadmap for agent-based computing. AgentLink report, available from *www.agentlink.org/roadmap*., 2003.
16. S. A. O'Malley and S. A. DeLoach. Determining when to use an agent-oriented software engineering. In *Proceedings of the Second International Workshop On Agent-Oriented Software Engineering (AOSE-2001)*, pages 188–205, Montreal, May 2001.

17. Lin Padgham. Design of multi agent systems. Tutorial notes, available from the author, July 2003.
18. Lin Padgham and Michael Winikoff. Prometheus: A methodology for developing intelligent agents. In *Third International Workshop on Agent-Oriented Software Engineering*, July 2002.
19. Lin Padgham and Michael Winikoff. Prometheus: A pragmatic methodology for engineering intelligent agents. In *Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies*, pages 97–108, Seattle, November 2002.
20. Lin Padgham and Michael Winikoff. *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley & sons, Ltd., 2004. ISBN 0-470-86120-7.
21. David Poutakidis, Lin Padgham, and Michael Winikoff. Debugging multi-agent systems using design artifacts: The case of interaction protocols. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'02)*, 2002.
22. Michael Prasse. Evaluation of object-oriented modelling languages: A comparison between OML and UML. In Martin Schader and Axel Korthaus, editors, *The Unified Modeling Language – Technical Aspects and Applications*, pages 58–75. Physica-Verlag, Heidelberg, 1998.
23. J. Rumbaugh. Notation notes: Principles for choosing notation. *Journal of Object-Oriented Programming (JOOP)*, 8(10):11–14, May 1996.
24. Onn Shehory and Arnon Sturm. Evaluation of modeling techniques for agent-based systems. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 624–631. ACM Press, May 2001.
25. A. Sturm and O. Shehory. Towards industrially applicable modeling technique for agent-based systems (poster). In *Proceedings of International Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, July 2002.
26. Arnon Sturm and Onn Shehory. A framework for evaluating agent-oriented methodologies. In *Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (at AAMAS03)*, July 2003.
27. Michael Winikoff and Lin Padgham. The prometheus methodology. In Federico Bergenti, Marie-Pierre Gleizes, and Franco Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems*, chapter 11. Kluwer Academic Publishing (New York), 2004.
28. B. Wood, R. Pethia, L.R. Gold, and R Firth. A guide to the assessment of software development methods. Technical Report 88-TR-8, Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA, 1988.
29. M. Wooldridge, N.R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the third international conference on Autonomous Agents (Agents-99)*, Seattle, WA, May 1999. ACM.
30. M. Wooldridge, N.R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3), 2000.
31. E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, 1995.

A Framework for Evaluating Agent-Oriented Methodologies

Arnon Sturm[1] and Onn Shehory[2]

[1] Technion - Israel Institute of Technology, Haifa 32000, Israel
sturm@techunix.technion.ac.il
[2] IBM Haifa Research Lab, Haifa University, Haifa 31905, Israel
onn@il.ibm.com

Abstract. Multiple agent-oriented methodologies has been introduced in recent years, although only partial evaluations of these have been offered. As a result, it is difficult to select a methodology for a specific project. Additionally, there are no means for determining what the advantages and drawbacks of each methodology are. To resolve these problems, we devise a framework for evaluating and comparing agent-oriented methodologies. This framework focuses on four major aspects of a methodology: concepts and properties, notations and modelling techniques, process and pragmatics. We demonstrate the usage of the suggested framework by evaluating the GAIA methodology. This evaluation identifies the strengths and the weaknesses of GAIA, thus exemplifying the capabilities of our framework.

1 Introduction

During the last decade, many methodologies for developing agent-based systems have been developed. A methodology is the set of guidelines for covering the whole lifecycle of system development both technically and managerially. A methodology, according to [10], should provide the following: a full lifecycle process; a comprehensive set of concepts and models; a full set of techniques (rules, guidelines, heuristics); a fully delineated set of deliverables; a modelling language; a set of metrics; quality assurance; coding (and other) standards; reuse advice; and guidelines for project management. The relationships between these components are shown in Figure 1. In that figure, we use the UML notations to depict the relationships between the components. As depicted in the figure, a methodology consists of a set of techniques, a modelling language and a lifecycle process. The set of techniques consists of metrics, quality assurance (QA) activities, a set of standards and tools. The modelling language comprises notations and a meta model. The lifecycle process consists of project management, a number of roles (e.g., an analyst or a designer), a number of procedures (e.g., how to move between development stages), and a number of deliverables (e.g., a design document, source code). In addition, Figure 1 shows that the tools should be based on the metamodel of the modelling technique and should represent the modelling technique's notations. The deliverables should use the modelling technique.

P. Giorgini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 94–109, 2004.

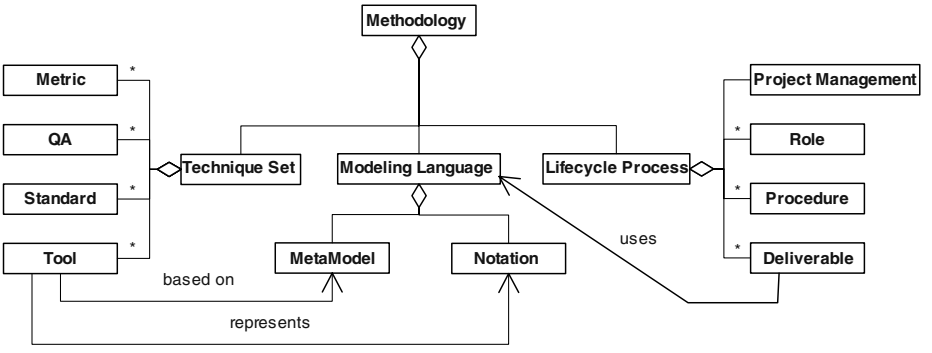

Fig. 1. The components of a methodology and the relationships among them.

There are more than two dozen agent-oriented methodologies. The multiplicity and variety of methodologies result in the following problems: (1) Industrial problem: selecting a methodology for developing an agent-based system/application becomes a non-trivial task, in particular for industrial developers who hold specific requirements and constraints; (2) Standards problem: multiple different methodologies are counter-productive for arriving at a standard. With no standard available, potential industrial adopters of agent technology refrain from using it; (3) Research problems: excessive efforts are spent on developing agent-oriented methodologies, at times producing overlapping results. Additionally, as a result of allocating resources to multiple methodologies, no methodology is allocated sufficient research resources to enable it to address all aspects and providing a fully-fledged agent-oriented methodology. In this paper, we provide a means for addressing these problems by supplying a framework for evaluating agent-oriented methodologies. This evaluation framework may be used by organizations to select a methodology for developing agent-based applications. It can also help researchers to examine the similarity and the differences among existing agent-oriented methodologies and to analyze the needed attributes of such methodologies. This can be done by identifying the extent to which the examined methodologies address the various aspects defined by the framework. Additionally, setting a scale for grading agent-oriented methodologies, and using the scale in conjunction with our framework, may result in a selection of the better methodologies, gradually reducing their number. This selection may eventually converge to a small set of the most fit agent-oriented methodologies, possibly leading to standardization.

Agent-oriented methodologies can be classified into two major classes: general-purpose methodologies and domain-specific methodologies. The framework provided in this paper is aimed for the evaluation of general-purpose methodologies, e.g., GAIA [20,22], Tropos [2] and MaSE [9]. Yet, evaluating these methodologies introduces several difficulties:

- Comparing methodologies is often difficult, because they might address different aspects or differ in their terminology. For example, they are based on different concepts – MaSE is based on software engineering concepts (such as state machines and components) and Tropos is based on knowledge level concepts (such as actors, goals, dependencies and plans).

- Some of the methodologies are influenced by a specific approach, e.g., BDI or OO.
- The completeness of various methodologies varies dramatically. For example, some provide only the process, some present graphical notations, while others integrate several aspects of a methodology (i.e., process, notations, guidelines etc.).

Since the agent-oriented paradigm can be considered as an evolution of the object-oriented paradigm, we considered the type of evaluations made for the object-oriented methodologies. Such evaluations have been discussed by many studies as indicated by [18]. These evaluations and comparisons suffer from a number of common flaws as follows:

- Using an inappropriate framework for performing the evaluation.
- There is no agreement on what a methodology is and of what it should consist.
- Sometimes there is no ranking for the support of a particular concept of a specific methodology. This leads to unsatisfactory results of the evaluation.
- In many cases the evaluation cannot be repeated.

A few evaluations of agent-oriented methodologies have been suggested. In [21], the authors set a list of questions that a methodology should address. However, neither evaluation nor a comparison has been performed using that set. Another study [4], suggests a framework for evaluating agent-oriented methodologies; however, the compared criteria refer only to the expressiveness of the methodologies and not to the wider set encompassed within the methodology definition. In [12], the author performs an evaluation of five agent-oriented methodologies; however, he refers only to some supported concepts such as organization design and cooperation and not to the broad set of attributes that constitute a complete methodology. In [14], the authors perform an evaluation of only the modelling part within a methodology. In [8] a comparison of agent-oriented methodologies is presented. The comparison is based on a survey [7], which is based on similar concepts to the framework suggested in this paper. In addition, the authors provide a structural comparison of the differences and commonalities of a number of methodologies using a crude (four value) evaluation scale. However, because methodologies usually address a much richer scale, analyzing them using a crude scale is rather restrictive and the expressiveness of the results is limited. In addition, the results provided in that research are biased, since the responders were the developers of the methodologies and students. In [5], the authors present a framework for evaluating agent-oriented methodologies, but it lacks in providing guidelines for evaluating the methodologies according to the criteria they present. Other studies that deal with evaluating agent-oriented methodologies compared two or three methodologies, yet mainly with respect to the expressiveness and the concepts supported by the methodology.

In this paper, we provide a comprehensive framework for evaluating and comparing agent-oriented methodologies. By comprehensive, we refer to the fact that the framework addresses a large set of aspects of an agent-oriented methodology that require analysis. This framework offers a well-defined, structured set of aspects that

an agent-oriented methodology should include. Other related studies refer only to a sub-set of the aspects that should be supported by a methodology. In addition, our framework provides a metric for grading evaluation criteria. The provided framework is a qualitative one, although it can be transformed into a quantitative one by borrowing the concepts from [4]. The suggested framework, although based on the framework suggested by [18], extends and modifies it to address the unique requirements of agent-oriented methodologies. The framework can be used for various evaluation techniques:

- Feature analysis – The evaluation is done by referring to the available resources.
- Survey – The evaluation is done by examining the results of the survey that is distributed among practitioners and researchers. As a result of the size of the population surveyed, these results may be statistically justified.
- Case studies – The evaluation is done by examining the results of case studies.
- Field experiments – The evaluation is done by examining the results of field experiments. By field experiments we refer to the assessment of a methodology in real projects and the control of experiment variables.

Due to lack of space we will not discuss the advantages and drawbacks of each technique. However, such a discussion can be found in [15].

The specific use of these techniques is determined mainly by resource constraints. In this paper, we perform the evaluation using the feature analysis technique to demonstrate its applicability and ease of use. An evaluation of this type can be easily performed by an organization because it can be performed internally within the organization and can be confined to a small group of evaluators. Yet, this technique is subjective and this is its major drawback. A survey technique (for example, using [7]) may reduce subjectivity, although it requires much more resources.

The paper is organized as follows. In Section 2 we introduce and define the evaluation framework. In Section 3 we perform an evaluation over a well-known methodology: GAIA and Section 4 concludes.

2 The Evaluation Framework

In this paper, we refer to a methodology as the entire set of guidelines and activities: a full lifecycle process; a comprehensive set of concepts and models; a full set of techniques (rules, guidelines, heuristics); a fully delineated set of deliverables; a modelling language; a set of metrics; quality assurance; coding (and other) standards; reuse advice; guidelines for project management. These are each associated with one of four major divisions: concepts and properties, notations and modelling techniques, process and pragmatics.

2.1 Concepts and Properties

A concept is an abstraction or a notion inferred or derived from specific instances within a problem domain. A property is a special capability or a characteristic. This

section deals with the question whether a methodology adheres to the basic notions (concepts and properties) of agents and multi-agent systems. In order to perform such an evaluation we need to define these concepts, since there is no agreement (yet) within the agent community regarding the basic concepts of agent-based systems. In this paper, we leverage on previous studies (e.g., [7, 14, 15, 17, 19]) and utilize notions defined there as a basis for our set of concepts. The following are the properties according to which an agent-oriented methodology should be evaluated:

1. **Autonomy:** is the ability of an agent to operate without supervision.
2. **Reactiveness:** is the ability of an agent to respond in a timely manner to changes in the environment.
3. **Proactiveness:** is the ability of an agent to pursue new goals.
4. **Sociality:** is the ability of an agent to interact with other agents by sending and receiving messages, routing these messages and understanding them. In addition, sociality refers to the ability of agents to form societies and be part of an organization.

In addition to the above properties, an evaluation of an agent-oriented methodology should examine whether the methodology supports the following building blocks along their notations and to what extent it does so.

1. **Agent** - A computer program that can accept tasks, can figure out which actions to perform in order to perform these tasks and can actually perform these actions without supervision. It is capable of performing a set of tasks and providing a set of services.
2. **Belief** - A fact that is believed to be true about the world.
3. **Desire** - A fact of which the current value is false and the agent (that owns the desire) would prefer that it be true. Desires within an entity may be contradictory. A widely used specialization of a desire is a goal. The set of goals within an agent should be consistent.
4. **Intention** - A fact that represents the way of realizing a desire.
5. **Message** - A means of exchanging facts or objects between entities.
6. **Norm** - A guideline that characterizes a society. An agent that wishes to be a member of the society is required to follow all of the norms within. A norm can be referred to as a rule.
7. **Organization** - A group of agents working together to achieve a common purpose. An organization consists of roles that characterize the agents that are members of the organization.
8. **Protocol** - An ordered set of messages that together define the admissible patterns of a particular type of interaction between entities.
9. **Role** - An abstract representation of an agent's function, service, or identification within a group.
10. **Service:** An interface that is supplied by an agent to the external world. It is a set of tasks that together offer some functional operation. A service may consist of other services.

11. **Society** - A collection of agents and organizations that collaborate to promote their individual goals.
12. **Task** - A piece of work that can be assigned to an agent or performed by it. It may be a function to be performed and may have time constraints. Sometimes referred to as an action.

2.2 Notations and Modelling Techniques

Notations are a technical system of symbols used to represent elements within a system. A modelling technique is a set of models that depict a system at different levels of abstraction and different system's aspects. This section deals with the properties to which methodology's notations and modelling techniques should adhere. The list of these properties is taken from [14].

1. **Accessibility:** is an attribute that refers to the ease, or the simplicity, of understanding and using a method. It enhances both experts' and novices' capabilities of using a new concept**.**
2. **Analyzability:** is a capability of checking the internal consistency or implications of models or to identify aspects that seem to be unclear, such as the interrelations among seemingly unrelated operations. This capability is usually supported by automatic tools.
3. **Complexity management** (abstraction)**:** is an ability to deal with various levels of abstraction (i.e., various levels of detail). Sometimes, high-level requirements are needed, while, in other situations, more detail is required. For example, examining the top level design of a multi-agent system, one would like to understand which agents are within the system, but not necessarily what their attributes and characterizations are. However, when concentrating on a specific task of an agent, the details are much more important than the system architecture.
4. **Executability** (and testability)**:** is a capability of performing a simulation or generating a prototype of at least some aspects of a specification. These would demonstrate possible behaviors of the system being modelled, and help developers determine whether the intended requirements have been expressed.
5. **Expressiveness** (and applicability to multiple domains)**:** is a capability of presenting system concepts that refers to:
   - the structure of the system;
   - the knowledge encapsulated within the system;
   - the system's ontology;
   - the data flow within the system;
   - the control flow within the system;
   - the concurrent activities within the system (and the agents)
   - the resource constraints within the system (e.g., time, CPU and memory);
   - the system's physical architecture;
   - the agents' mobility;

- the interaction of the system with external systems; and
- the user interface definitions.

6. **Modularity** (incrementality)**:** is the ability to specify a system in an iterative incremental manner. That is, when new requirements are added it should not affect the existing specifications but may use them.
7. **Preciseness:** is an attribute of disambiguity. It allows users to avoid misinterpretation of the existing models.

2.3 Process

A development process is a series of actions, changes and functions that, when performed, result in a working computerized system. This section deals with the process development aspect of a methodology. This aspect is evaluated using the following issues:

1. **Development context:** specifies whether a methodology is useful in creating new software, reengineering or reverse engineering existing software, prototyping or designing for or with reuse components.
2. **Lifecycle coverage:** Lifecycle coverage of a particular methodology involves ascertaining what elements of software development are dealt with within the methodology. Each methodology may have elements that are useful to several stages of the development life cycle. In this paper, the lifecycle stages are defined as follows:
   - Requirements' gathering is the stage of the lifecycle in which the specification (usually in free text) of the necessities from the system is done.
   - Analysis is the stage of the lifecycle that describes the outwardly observable characteristics of the system, e.g., functionality, performance and capacity.
   - Design is the stage of the lifecycle that defines the way in which the system will accomplish its requirements. The models defined in the analysis stage are either refined, or transformed, into design models that depict the logical and the physical nature of the software product.
   - Implementation is the stage of the lifecycle that converts the developed design models into software executable within the system environment. This either involves the hand coding of program units, the automated generation of such code or the assembly of already built and tested reusable code components from an in-house reusability library.
   - Testing focuses on ensuring that each deliverable from each stage conforms to, and addresses, the stated user requirements.

Having the development stages defined is not sufficient for using a methodology. A methodology should further elaborate the activities within the development lifecycle, in order to provide the user of the methodology with the means of using it prop-

erly and efficiently. Providing a detailed description of the various activities during the development lifecycle would enhance the appropriate use a methodology and increase its acceptability as a well-formed engineering approach. Hence, we suggest examining the process in a more detailed way. These details can be provided by answering the following questions regarding the evaluated methodology:

1. What are the activities within each stage of a methodology? For example, an activity can be the identification of a role, a task etc. It may consist of heuristics or guidelines helping the developer to achieve his/her goals (in developing the system).
2. What deliverables are generated by the process? This question refers mainly to the documentation. For example, what models are specified and can be delivered to the customer. Another example is whether an acceptance testing plan is required and when it is required.
3. Does the process provide for verification? This question checks whether a methodology has rules for verifying adherence of its deliverables to the requirements.
4. Does the process provide for validation? This question checks whether a methodology has rules for validating that the deliverables of one stage are consistent with its preceding stage.
5. Are quality assurance guidelines supplied?
6. Are there guidelines for project management?

2.4 Pragmatics

Pragmatics refers to dealing with practical aspects of deploying and using a methodology. This section deals with pragmatics of adopting the methodology for a project or within an organization. In particular, the framework suggests examining the following issues:

1. **Resources:** What resources are available in order to support the methodology? Is a textbook available? Are users' groups established? Is training and consulting offered by the vendor and/or third parties? In addition, are automated tools (CASE tools) available in support of the methodology (e.g., graphical editors, code generators, and checkers)? This issue should be examined in order to enable a project/organization aiming at adopting a methodology to check the resources (in terms of training and budget) required and the alternatives for acquiring these.
2. **Required expertise:** What is the required background of those learning the methodology? A distinguishing characteristic of many methodologies is the level of mathematical sophistication required to fully exploit the methodology. Does the methodology assume knowledge in some discipline? This issue should be examined in order to enable a project/organization aiming at adopting a methodology to check whether the qualifications required for using the methodology are met by the candidate users.

3. **Language (paradigm and architecture) suitability:** Is the methodology targetted at a particular implementation language? That is, is the methodology based on concepts from a specific architecture or a programming language? For example, a methodology may be limited to BDI-based applications; it may be oriented towards a specific object-oriented language. This issue should be examined to check whether a methodology adheres to the organization/project infrastructure and knowledge.
4. **Domain applicability:** Is the use of the methodology suitable for a particular application domain (e.g., real-time and information systems)? This issue should be examined to check whether the methodology adheres to the intended problem domain.
5. **Scalability:** Can the methodology, or subsets thereof, be used to handle various application sizes? For example, can it provide a lightweight version for simpler problems? This issue should be examined to check whether the methodology is appropriate for handling the intended scale of applications within the project/organization.

2.5 Metric

To enable ranking the properties examined in the evaluation process, we propose a scale of 1 to 7 as follows:

1. Indicates that the methodology does not address the property.
2. Indicates that the methodology refers to the property but no details are provided.
3. Indicates that the methodology addresses the property to a limited extent. That is, many issues that are related to the specific property are not addressed.
4. Indicates that the methodology addresses the property, yet some major issues are lacking.
5. Indicates that the methodology addresses the property, however, it lacks one or two major issues related to the specific property.
6. Indicates that the methodology addresses the property with minor deficiencies.
7. Indicates that the methodology fully addresses the property.

In summary, in this section we provided a framework for evaluating agent-oriented methodologies. We divide that framework into four divisions of concepts and properties, notations and modelling techniques, process, and pragmatics. In the next section we demonstrate the use of that framework.

3 Evaluating GAIA

In this section we evaluate GAIA according to the framework presented in Section 2. We are fully aware of studies that extend GAIA in various aspects such as expressiveness [11] and implementation [13]. However, in the evaluation we performed, we refer only to [20] and [22], as they were written by the methodology designers.

3.1 Concepts and Properties

Below, we examine the extent to which GAIA addresses the concepts and the properties suggested by the evaluation framework. GAIA deals with all of the properties suggested, but lacks in depicting mental states of an agent.

1. **Autonomy:** In GAIA the autonomy is expressed by the fact that the role encapsulates its functionality (i.e., it is responsible for it). This functionality is internal and is not affected by the environment, thus representing the role's autonomy. In addition, in GAIA there is an option to model alternative computational paths, which gives the role (and agents that consist of this role) autonomy in making decisions. The ranking grade is 7.
2. **Reactiveness:** In GAIA the reactiveness is expressed by the liveness properties within the role's responsibilities. The ranking grade is 7.
3. **Proactiveness:** In GAIA the proactiveness is expressed by the liveness properties within the role's responsibilities. The ranking grade is 7.
4. **Sociality:** in GAIA the sociality is expressed within the acquaintance model that defines the communication links among agent types. Further, some sociality aspects can be expressed using the organizational structure and rules. Yet, there is no explicit specification of relationships between organizations and roles, and societies, within an MAS. The ranking grade is 4.

Table 1. The coverage of the framework building blocks within GAIA.

Framework building block	GAIA concepts
Agent	Agent type
Belief	
Desire	
Intention	
Message	Protocol
Norm	Organizational rule
Organization	System
Protocol	Protocol
Role	Role
Service	
Society	System
Task	Activity, Responsibility

GAIA captures the MAS as a society or an organization. It defines a few abstract concepts as follows: a role, a permission, a responsibility, a protocol and an activity. A role is a definition of functionality within a social group. A permission identifies the resources a role can access. A responsibility is the actual definition of the role functionality. A protocol specifies the way by which a role interacts with another role and an activity is a private action of the role. In addition, GAIA defines a few concrete concepts as follows: an agent type and a service. An agent type is a set of roles and a service is a coherent block of activity, which is defined by pre- and post condi-

tions, inputs, and outputs. Furthermore, GAIA defines a term of an organizational rule to capture the notion of general system constraints.

Examining the coverage of the framework building blocks by GAIA, we found that GAIA addresses most of them, as seen in Table 1. However, the BDI concepts and the knowledge representation as well are not dealt with within GAIA. In addition, GAIA does not support the definition of a service. The ranking grade is 5.

3.2 Notations and Modelling Techniques

Following, we examine the extent to which GAIA addresses the notations and modelling techniques' properties suggested by the evaluation framework. GAIA has room for improvements with regards to these properties. In addition, GAIA does not define its entire set of notations.

1. **Accessibility:** GAIA models are basically easy to understand and use. Yet, the behavior of the system is introduced via a set of logic expressions that might introduce difficulties to those who would like to understand it. The ranking grade is 5.
2. **Analyzability:** this issue is not dealt with within GAIA. The ranking grade is 1.
3. **Complexity management:** in GAIA, there is no hierarchical presentation or any other mechanism for complexity management. The system description is flat. The ranking grade is 1.
4. **Executability:** this issue is not dealt with within GAIA. The ranking grade is 1.
5. **Expressiveness:** GAIA is expressive and can handle a large variety of systems due to its generic structure. However, GAIA is mostly suitable for small and medium scale systems. This is because of its flatness, which limits the ability to model a large amount of details. In the following we present our analysis regarding the expressiveness of GAIA according to the properties defined in the previous section:
   - the structure of the system is not presented explicitly;
   - the knowledge encapsulated within the system is not presented explicitly;
   - the system's ontology is not dealt with;
   - the data flow within the system is depicted using textual specifications (via the dot and square brackets operators);
   - the control flow within the system is not presented explicitly;
   - concurrent activities within the system (and the agents) are not presented explicitly;
   - the resource constraints within the system (e.g., time, CPU and memory) are partially specified using the permissions within GAIA;
   - the system's physical architecture is not dealt with;
   - the agents' mobility is not dealt with;
   - the interaction of the system with external systems is not presented explicitly;
   - and the user interface definitions is not dealt with.

   The ranking grade is 3.

6. **Modularity:** GAIA is mostly modular due to its design with some building blocks such as roles, protocols, activities and agent types. In GAIA, one can assign new roles to agents and remove roles with no effect on the internal model of the roles. However, changes within the protocol might cause changes within the internal structure of the role. These result in changes in permissions of the role, hence limiting the modularity of GAIA. The ranking grade is 4.
7. **Preciseness:** the liveness and safety properties, which are used for depicting the functionality of a role in a formal way (i.e., for each symbol and notation there is a clear meaning and interpretation), make GAIA accurate and prevents misinterpretation of the modelled functionality. The symbols and notations of each of the other GAIA models have a clear meaning as well. The ranking grade is 7.

3.3 Process

Below, we examine the extent to which GAIA addresses the development process properties suggested by the framework. GAIA deals only with some aspects of the development process. With respect to the lifecycle coverage, it handles the analysis and design stages.

1. **Development context:** GAIA is adequate for the following development contexts: it can be used in creating new software, reengineering and designing systems with reuse components. However, GAIA does not support classical reverse engineering (from code to a model), since it does not address the implementation aspects. For the same reason, it cannot be used for prototyping (especially, a rapid one). The ranking grade is 5.
2. **Lifecycle coverage:** lifecycle coverage of GAIA is very limited. It refers only to the analysis and the design stages within the development lifecycle. We found that fact a drawback of GAIA as a methodology, since it would require developers of MAS to adjust the GAIA-based design to the concepts of the target programming language. For example, one may translate the GAIA analysis and design results to UML notations and than use an object-oriented language for implementation. The ranking grade is 3.
3. **Stages' activities within the methodology:** The analysis phase within GAIA includes the identification of the overall goals of the organization and its expected global behavior, the basic skills required by the organization and the basic interactions required for the exploitation of these skills, and the rules that the organization should respect or/and enforce in its global behavior. The design phase within GAIA includes the definition of the organizational structure, the refinement of the roles and the interaction from the analysis phase, the exploitation of well-known organizational patterns, the definition of the agent types that make up the system and assigning roles to them, and the definition of the services that are required for realizing the roles. Overall, GAIA provides only a few guidelines for performing the aforementioned activities. The ranking grade is 4.

4. **Methodology deliverables:** each stage supported by GAIA has its own deliverables. The outcomes of the analysis phase are a preliminary role model, a preliminary interaction model and a set of organizational rules. The outcomes of the design phase are a role model, an interaction model, an organizational structure, a set of organizational rules, a service model and an agent model. The ranking grade is 7.
5. **Verification and validation:** this issue is not dealt with within GAIA. The ranking grade is 1.
6. **Quality assurance:** this issue is not dealt with within GAIA. The ranking grade is 1.
7. **Project management guidelines:** this issue is not dealt with within GAIA. The ranking grade is 1.

3.4 Pragmatics

Following, we examine the extent to which GAIA addresses the pragmatics properties suggested by the evaluation framework. Apparently, GAIA lacks in addressing some of these properties.

1. **Resources:** Although GAIA is well known, there are not much available materials on it (except for the two cited papers). There are no users' groups, nor any training or consulting services offered. Additionally, GAIA does not provide automated tools. The ranking grade is 3.
2. **Required expertise:** GAIA requires a solid background and knowledge in logic and temporal logic. This causes a reduction in its accessibility since many developers do not know or do not want to get familiar with logic (and formal methods). The ranking grade is 2.
3. **Language suitability:** GAIA is not targeted at a specific language. It does not refer to the implementation issues. Thus, the specification made using GAIA can be implemented in any language. The ranking grade is 7.
4. **Domain applicability:** GAIA, as determined by its developers, is suitable to develop applications with the following characteristics: agents are coarse-grained computational systems, a global goal exists, agents are heterogeneous, the organizational structure is static, the abilities of the agents are static and the number of agent types is comparatively small. Yet, GAIA is not suitable for developing applications with dynamic characteristics such as goals generation and changing organizational structure. The ranking grade is 4.
5. **Scalability:** GAIA does not support the use of subsets thereof for system development. Yet, due to its simple structure, it may fit different application sizes. The ranking grade is 4.

3.5 Evaluation Summary

In this section we summarize the evaluation of GAIA. This evaluation demonstrates the use of the proposed framework and the way in which it identifies the strengths

and the weaknesses of a methodology. Examining the concepts and properties (as defined by the framework) supported by GAIA, we found that GAIA addresses these to a satisfactory level. Examining the notations and modelling techniques provided by GAIA, we found that it addresses these to a limited extent, mainly due to lack of support in software engineering principles and an insufficient expressiveness of implementation-oriented issues. Examining the process elements provided by GAIA, we found that GAIA provides little details on it; hence, further enhancements are required. Finally, examining the pragmatics supported by GAIA, we found that GAIA has a solid theoretical basis but it lacks means to enforce it. Given these results, it seems that there was no intention for GAIA to support stages other than the analysis and design stages and, as a result, it lacks in support for other stages.

4 Conclusions

In this paper, we propose a framework for evaluating and comparing agent-oriented methodologies. The framework examines the various aspects of a methodology: concepts and properties, notations and modelling techniques, process and pragmatics.

This framework addresses the problem of evaluating and comparing methodologies. It can be utilized for selecting a methodology for developing an agent-based application. It can additionally be utilized for identifying the advantages and weaknesses of existing agent-oriented methodologies. Identifying these can promote the improvement of such methodologies. Improvement of these may advance the acceptability of agent technology by introducing a mature, well-structured engineering approach.

We demonstrate the use of the framework using a feature analysis technique by performing an evaluation of GAIA – an evolving agent-oriented methodology. GAIA is justifiably considered an advanced agent-oriented methodology. Nevertheless, as our study shows, there are several aspects in which the GAIA methodology can be improved, to provide an industry grade methodology. By detecting the shortcomings (and providing the details) of one of the most advanced agent-oriented methodologies, we show that our framework is applicable: it points at the weaknesses of a methodology and thus can promote its improvement. Our results are reinforced by [8], where another, slightly different, evaluation framework is used. In that evaluation, GAIA's grades relevant to concepts were leaning to medium (in a discrete scale of low, medium and high). With respect to the modelling language grades, that evaluation shows that GAIA addresses only some of the criteria. Referring to the process grades, that evaluation shows that GAIA handles only the requirements analysis and the architectural design stages. In addition, that evaluation shows that GAIA lacks in its pragmatics. Our results are also strengthened by the work in [5] and its evaluation of GAIA [6]. That evaluation indicates that GAIA handles only the analysis and design stages, it lacks supportive tools, it lacks documentation and suffers from expressiveness problems. The aforementioned studies all arrive at results similar to ours. This implies that the quality of the analysis performed using our framework is as good as others' with respect to the subset of aspects analyzed by

them. However, our framework analyzes a richer set of criteria, thus better covering the requirements from a methodology. Although we presented the framework and the feature analysis technique as well-structured and easy to use, we are fully aware of its subjectivity. That is, the ranking grades may vary across evaluators. However, we believe that the overall evaluations resulting from using the proposed framework by several evaluators will be similar due to the well-defined properties and the ranking scale.

Further research is required to evaluate the suggested framework. It may be evaluated with respect to several criteria: accessibility, coverage, adaptability. Accessibility refers to the ease of learning and using the framework; coverage refers to the extent to which the framework addresses the needs of methodologies' evaluation; adaptability refers to the ability to modify and adjust the framework to evaluate domain-specific agent-oriented methodologies. Another research direction could be a comparative evaluation of agent-oriented methodologies, utilizing the proposed framework.

References

1. Bayer P., Svantesson M., "Comparison of Agent-Oriented Methodologies Analysis and Design, MAS-CommonKADS versus Gaia", Blekinge Institute of Technology, Student Workshop on Agent Programming, www.soclab.bth.se/communities/bitswap/ bayer2001a.pdf, 2001.
2. Bresciani P., Giorgini P., Giunchiglia F., Mylopoulos J., Perini A., "TROPOS: An Agent-Oriented Software Development Methodology". Accepted to the Journal of Autonomous Agents and Multi-Agent Systems, 2003.
3. Brinkkemper S., Hong S., van den Goor G., "A formal approach to the comparison of object-oriented analysis and design methodologies", in Proc.of the Twenty-Sixth Hawaii Intl. Conf. on , Vol. 4 , pp. 689-698, Jan 1993.
4. Cernuzzi L., Rossi G., "On the Evaluation of Agent Oriented Methodologies", in Proc. of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies, November 2002.
5. Cuesta P., Gómez A., González J. C., and Rodríguez F. J., A Framework for Evaluation of Agent Oriented Methodologies, The Conference of the Spanish Association for Artificial Intelligence (CAEPIA), 2003.
6. Cuesta P., Gómez A., González J. C., and Rodríguez F. J., Evaluating GAIA, http://ma.ei.uvigo.es/framework/Framework_Gaia.pdf, 2003.
7. Dam K. H., Winikoff M., Survey on Agent-Oriented Methodologies, http://yallara.cs.rmit.edu.au/~kdam/Questionnaire/Questionnaire.html, 2002.
8. Dam K. H, Evaluating Agent-Oriented Software Engineering Methodologies – M.Sc. Thesis, 2003.
9. DeLoach S. A., Wood M. F., Sparkman C. H., "Multiagent Systems Engineering", The Intl. Jour. of SE and KE, Vol. 11, No. 3, June 2001.
10. Graham I., Henderson-Sellers B., Younessi H., "The OPEN Process Specification", Addison-Wesley, 1997.
11. Juan T., Pearce A., Sterling L., "ROADMAP: Extending the GAIA Methodology for Complex Open Systems", in Proc. of AAMAS'02, pp. 3-10, July 2002.

12. Kumar M., “Contrast and comparison of five major Agent Oriented Software Engineering (AOSE) methodologies”, http://students.jmc.ksu.edu/grad/madhukar/www/professional/aosepaper.pdf, 2002.
13. Moraitis P., Petraki E., Spanoudakis N. I., “Engineering JADE Agents with the Gaia Methodology “, Proc. Int. Workshop on Agents and Software Engineering, 2002.
14. Shehory O., Sturm A., “Evaluation of modelling techniques for agent-based systems”, Agents 2001, pp. 624-631, 2001.
15. Siau K. and Rossi M., "Evaluation of Information Modelling Methods – A Review", in Proc. 31 Annual Hawaii International Conference on System Science, pp. 314-322, January 1998.
16. Sturm A., Dori D., Shehory O., “Single-Model Method for Specifying Multi-Agent Systems”, Proc. Of AAMAS’03, 2003.
17. Tran Q. N., Low G., Williams M. A., Software Engineering Methodology for Developing Multi-Agent Systems - A Survey, http://129.94.244.146/personal/numi+tran/ surveyq.nsf/survey, 2003.
18. The Object Agency, “A Comparison of Object-Oriented Development Methodologies”, Technical report, http://www.toa.com/smnn?mcr.html, 1995.
19. Winikoff M., Padgham L., and Harland J., “Simplifying the Development of Intelligent Agents”, Proc of the 14th Australian Joint Conference on Artificial Intelligence (AI'01), pages 557-568, 2001.
20. Wooldridge M., Jennings N. R., Kinny D., "The Gaia Methodology for Agent-Oriented Analysis and Design", Journal of Autonomous Agents and Multi Agent Systems, Vol. 3, No. 3, pp. 285-312, March 2000.
21. Yu E., L. Cysneiros M., "Agent-Oriented Methodologies – Towards A Challenge Exemplar", 4th Intl. Workshop on Agent-Oriented Information Systems (AOIS’02), May 2002.
22. Zambonelli F., Jennings N., Wooldridge M., “Organizational Rules as an Abstraction for the Analysis and Design of Multiagent Systems”, Intl. Jour. of SE and KE, Vol. 11, No. 4, pp. 303-328, April 2001.

Towards Reuse in Agent Oriented Information Systems: The Importance of Being Purposive

Simon Goss[1,2], Clint Heinze[1,2], Michael Papasimeon[1,2], Adrian Pearce[1], and Leon Sterling[1]

[1] Intelligent Agent Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne
Parkville, VIC 3010, Australia
{clint,michp,pearce,leon}@cs.mu.oz.au
[2] Air Operations Division
System Sciences Laboratory
Defence Science and Technology Organisation
506 Lorimer Street, Fishermans Bend, VIC 3207, Australia
simon.goss@dsto.defence.gov.au

Abstract. The emergence of large information systems has pushed software specification into the area of business modelling to adequately capture and consider business requirements. At the same time, there has been a move toward techniques for specifying the behaviours of and the knowledge associated with intelligent agents as these are increasingly found as important components of those information systems. This paper presents four software models useful for specifying the requirements of an agent oriented information system. Adopting a similar notation for each model smooths the transition between models. It will be shown that it is in the relationships between these models there is scope for capturing *purposive descriptions* that facilitate reuse at various levels. A commentary on the importance of an explicit representation of the purpose for which a software component is intended is provided followed by an example, from the domain of military simulation, that illustrates the model and its application. The aim of this paper is to present a modelling approach that unifies business models, use case models, agent behavioural models and domain models, for the purpose of specifying an agent oriented information system.

1 Introduction

Many years of software engineering theory and practice have demonstrated the importance of developing software with well defined, precise, concise, unambiguous and documented requirements. The process of requirements specification and management, like in all engineering disciplines becomes increasingly important the larger the scale and scope of the software system being developed. High quality requirements specification and management techniques is particularly

P. Giorgini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 110–125, 2004.

relevant to the construction of modern information systems. This isn't only because modern information systems are historically some of the most complex systems developed by humans, but also because they are critical in the running and operations of many businesses.

Traditional approaches to requirements specifications for such information systems have included documents outlining the functional and non functional requirements of a system. These usually took the form of numbered requirements, laid out as linear statements, describing what the proposed system shall or shall not do. As information systems became more complex, these approaches were insufficient in modelling the requirements in a manner that facilitated the tasks of the client, the software analyst, designers and testers. Over the years such techniques have been supplemented and complemented with additional methods. For example, formal mathematical methods have been used to unambiguously specify requirements for complex systems, especially those with safety, security or mission critical constraints. In recent years there has been a shift in software requirements specification from *what* the system needs to do, to one that answers the question of *why*. To understand the complex requirements of information system, *what* a system does must be understood, but also *why* it does it. There has been a shift to a purposive perspective; the purpose for the development of a particular information system. Understanding its purpose helps to better develop the requirements of the system. For example, when specifying functionality there has been a shift from simply stating the functions that a system can perform, to one where the user and other actors become central in the specification. The functionality of the information system is presented from the perspective of the user and the tasks he or she needs to achieve in using the system.

In order to support reuse however, the purpose of the information system must be explicitly modelled through the software development lifecycle. Too often when going from requirements to design, the purpose of the system is implicit; that is, it is in the head of the designer. It is only when the purpose of a system, its components and its requirements are explicit that an adequate assessment of the potential for reuse can be made.

2 Requirements Modelling

2.1 Use Case Modelling

Use case modelling is a good example of a requirements specification technique which has a purposive perspective. The trend to such approaches is evident by the important and central role which use case modelling plays in the Unified Modeling Language (UML) [13] and its associated methodologies [7]. A use case model of an information system puts the requirements and the purpose of the system in context. Not only do we see the external interactions of the system through the explicit representation of actors such as users, and other software and hardware systems, but also the use cases describe the functionality of the system in terms of the purposes for which the actors use the system. The use case model becomes central in the specification of a modern information system and complements traditional approaches to requirements specification.

2.2 Business Modelling

A business model helps to define the actors, business workers and business processes that are present in an organisation. Formally defining the processes an organisation uses not only helps from a business perspective but also helps identify business processes which can be automated or that are highly dependent on certain information systems. More importantly however, a business model puts software use case models and software requirements specifications in the context of the activities the enterprise/business or organisation needs to undertake. The specification for the software is no longer viewed in isolation, but it is viewed in terms of the purpose of the software within the organisation. The development and use of a particular information system must satisfy a particular business need. Explicitly developing business and use case models in addition to traditional software requirements specifications ensures that the purpose of an information system is explicitly stated in terms the business requirements of an organisation.

A recent trend in the business process modelling community has been to adopt the notational conventions of the software modelling community. This is particularly true for methodologies such as the Unified Modelling Language (UML) where requirements specification notations such as use case diagrams have been adopted for capturing business requirements (Figure 1). The reason for this is that both information systems and businesses can be considered as complex systems with internal and external actors that need to undertake some activities (use cases). Whereas in traditional use case modelling we have use cases representing the functional requirements of the system; and we have actors, external entities that interact with the system via use cases. In business process modelling, we deal with business use cases representing the different processes that a business can undertake. Interacting with the business use cases we have external business actors representing the external people, organisations and systems with which a business may interact. These may be clients, other businesses or even other information systems such as external databases. These types of techniques have resulted in initiatives such as the specification of business modelling extensions to UML appearing as a standard UML extension profile; the adopting of Business Modelling as an import process in the Unified Process; and the adoption and custom extension of UML to business modelling [10].

2.3 Agent Behavioural Modelling

As information systems become more complex, some enterprises and organisations have begun to move to agents as either technology enablers or to more agent oriented software engineering methodologies as a means of abstraction to handle the complexity in these systems. As a result, not only is the specification of the system from an external perspective (i.e. use case modelling) important, but we must also try to understand the system from the point of view of the agents in the system. This agent model is the specification of the behaviour required of the agents in the system. From one perspective, this is an internal view

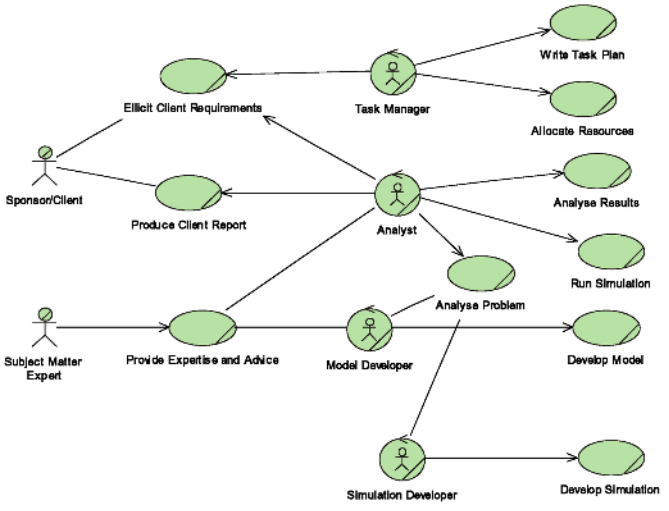

Fig. 1. A diagrammatic representation of part of a business model for Air Operations Analysis branch using UML. The sponsor/client are air force officers that will make use of the advice provided from by the branch. The subject matter experts (SME) are typically the operational crews of the aircraft and the engineers experienced in the technical aspects of the on-board systems.

of the information system not provided by the standard use case and business modelling approaches. From another perspective, it is still an external perspective. The behaviour of the agents is specified in the context of their interactions with their environment. For example, notions of internal agent mental states are not explored.

Agent behavioural modelling can use the same UML notation used for use case and business modelling. Just as use case and business modelling should not be seen as exclusive alternatives to other methods of requirements specification, similarly agent use case modelling should be viewed as a complementary technique. It should be used as an additional view into the requirements of a complex system that is amenable to an agent oriented perspective. Not all systems are amenable to such a view. However, there are benefits to be obtained in taking such an approach, especially for systems that are inherently agent oriented.

2.4 Domain Modelling

Because information systems are embedded in real business environments, the specification of those systems is contextually dependent upon knowledge of the real world. Managing, representing and modelling this knowledge assists the requirements engineering processes. Defining the behaviours that agents can undertake requires an understanding of the domain in which they operate. A domain model helps to articulate and communicate information about the domain in

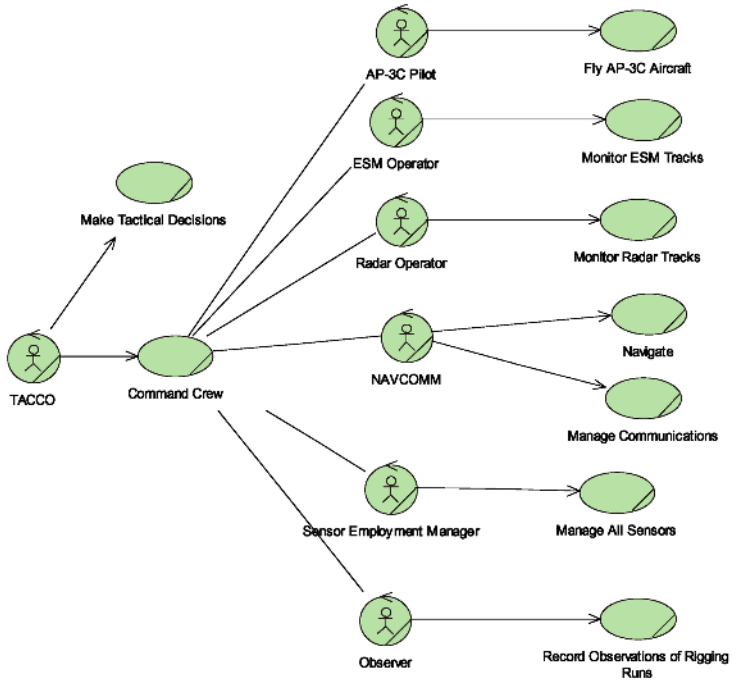

Fig. 2. Representation of a partial domain model in UML. This diagram shows the roles and relationships between the members of the crew on-board an AP-3C aircraft. This domain model is a store of knowledge that can be referenced to construct the agent models.

which the information system is being built in. This is especially true for newer or more complex systems where the software engineers and other stakeholders in the system can use a model of the domain as a common vocabulary in order to reach a shared understanding. A common notation is also important from this perspective. Therefore to maintain commonality and consistency between the different models, the UML should be adopted for domain modelling as well (Figure 2). In [6], examples of how to capture and represent domain knowledge using UML are shown.

3 Example Applications

In the previous section, use case, business, agent and domain models were discussed without specific reference to examples as how these models might be used. This section presents three brief examples of how the four models might be used in certain types of information systems. This is followed by a more detailed use case in the following section.

3.1 Air Combat Simulation

The Australian Defence Science and Technology Organisation (DSTO) conducts operational analysis to provide advice on the acquisition and tactical use of airborne systems such as fighter aircraft for the Australian Defence Force (ADF). One of the techniques that is used is that of air combat simulation. Large simulation of air combat are developed to study particular tactics or equipment. These simulation are completely constructive; there are no humans in the loop. Instead, there are virtual fighter pilots flying the virtual combat aircraft. The virtual pilots are developed using agent technology to represent the decision making, reasoning and tactical capabilities of real fighter pilots.

For some applications however, the agents are removed and are replaced with human operators (so called "Human-In-The-Loop" or "HiL" simulation). In these cases the models representing the physical world do not distinguish between being operated by humans or agents. Although there is a design and implementation interfacing issue, the behavioural specification for a human or a computational agent in such a system is identical. In fact, the construction of these type of simulation systems is simplified when a distinction is not made and the system is built in a way to accommodate agents or humans (that is a synthetic environment that is agnostic as to whether the intelligent entities operating in it are virtual (computational) or real. These findings are presented by Heinze et al. in [3].

In developing the requirements for such a system, a number of things must be considered. Obviously, the functionality available to analyst conducting a study is important. This includes the ability to specify, run, analyse and visualise a scenario. It is here that a use case model of the simulation system is needed. Because running these simulations is important to the core business of DSTO's Air Operations Division, a business model of the division is needed to explicitly show where air combat simulation fits in the overall process of providing advice to the ADF. The use case model describes the functionality of the system, primarily from the perspective of the main users who happen to be air combat analysts. However, this information isn't enough to specify what actually goes on in the simulation. The interesting behaviour is that which is performed by the pilot agents. Therefore an agent model is needed to specify the behaviours that the pilots should undertake in a particular simulation. Finally, the area of air operations is difficult to understand for the non-expert. An understanding of the domain is critical for the specification of the required behaviours. An understanding of the air combat domain calls for an explicit domain model.

Use Case Model. Actors include the analysts who use the simulation to obtain results and possibly external databases with which the simulation tool may interact. The main use cases involve analysts being able to specify air combat scenarios, run and control a simulation and then be able to visualise and analyse the results.

Business Model. This is a business model of an organisation performing analysis for the Australian Defence Force. Business workers include analysts, managers, modellers, subject matter experts and software engineers developing the simulations. External business actors include clients in the Australian Defence Force and external subject matter experts. Business use cases describe the major activities of the

organisation including the running of simulations in order to perform analysis to provide advice. This business model puts in context how the simulation software mentioned above is used in the organisation.

Agent Model. The air combat simulation constructed makes use of intelligent agents to represent human reasoning and tactical decision making of military operators in the scenarios being considered. These include fighter pilots, mission commanders, sensor operators etc. The agent use case model is a behavioural specification of the activities and tasks performed by the agents in the synthetic environment represented in the air combat simulation.

Domain Model. This is a model of the domain of air combat. The actors represent the people involved in air combat operations such as pilots and fighter controllers. The model specifies the activities that are undertaken in air combat operations and are used to help specify the behaviour in the agent model.

3.2 Game Development

Consider a game development company working on a game for a publisher. In the game specification document (also known as the *game design* in the industry), a use case model can be used to identify the activities that the game player can undertake in the game. The business model describes the game development organisation, not only internally (such as the relationship between the development staff such as engineers and artists) but also externally, with publishers and other organisations.

Similar to the air combat example in the previous section, being able to specify the behaviour of the game from the user's perspective only specifies the system partially. The behaviour of the characters (or agents) in the game need to be described. This is where the agent model comes in. Finally, if the game is set in a particularly complex fictitious (or simulated real) world then a domain model is required to lay out the boundaries and possibilities of the virtual world's domain.

Use Case Model. When specifying a use case model of a video game the primary actor is the game player. The use case model should describe the activities the player can undertake while using the game software, both in and out game elements. A game development may also choose to create use case models for other internal software tools used to develop the game. For example, an artist might make use of texturing tool and a level designer might use a level editor.

Business Model. The business model describes how the game development company operates and develops games. The primary business workers will include project managers, producers, artists, software engineers, game designers, sound engineers, programmers and testers. These business workers are involved in specific activities as part of the overall game development process. For example, a sound engineer might be involved in a number of use cases such as creating sounds effects for a game. The business model should also include external business workers which the game company interacts with such as the publisher, venture capitalists, the player community etc.

Agent Model. An agent use case model in a game development project will be used as a means to describe the behaviour of the characters (or other intelligent entities)

in a game. It should contain detailed information of how the characters interact with the virtual game world, other characters and the human players playing the game.

Domain Model. Most games have a background story that may be considered the domain in which the game lives in. This is more specific than a game genre. This model should contain information about the fictitious world developed for the game. It is the domain by which the character behaviour is constrained.

3.3 Online Retailing

Consider a web based store selling varying types of objects. There are many examples available today (books, food, software, videos etc). Critical to the operation of this company is the online information system. Therefore, the company must be able to specify how a customer should be able to interact and purchase items on the system via his or her web browser. This first step can be specified using a use case model. The business model shows the bigger picture of how the online information system fits into the overall running of the business such as the relationship with suppliers, inventory and stock control.

If the information system is sufficiently complex and offers value added features to the client that are implemented using agents, an agent use case model may be required. This may include agents for making recommendations, bargain finding agents, reminder agents and so forth. A use case model will be needed to describe how the agents behave in the context of this particular information system. When exploring different options for the agents to be made available on the system to the customers, a domain model may be built to help understand the particular domain being explored. For example, a domain model might be built to understand the concepts involved in bargain hunting. The agent model might then only describe a subset of these for inclusion in the final system.

Use Case Model. This model would describe the functionality of the online web store as well as the back end of the retail system. Thus, in addition to the obvious actors such as the customers who visit the web site, the use case model might show interactions with a catalogue database, an inventory system, a purchasing system etc. The model should describe all the different types of activities the customer can undertake via the web-site, as well as the general functionality of the system.

Business Model. The model describes how the online retailer operates the business, what the external relationships with suppliers, customers are etc. The internal relationships indicate perhaps the activities of the programmers who maintain the online presence.

Agent Model. In some online retailers, agents are used to provide an enhanced experience to the customer. These might include agents that manage a customer's wish-list, current shopping basket as well as agents that recommend purchases and remind the customer about special events like up-coming birthdays, as well as agents that search the internet to find the best bargain for the product the customer is looking for. The agent model should describe how these agents behave and how they interact with the user.

Domain Model. This model describes the domain in which the online retail agents operate. For example, it might describe the domain, rules and practices of bargain hunting or recommendation making.

4 Supporting Reuse

Reuse is supported by each of the four models developed in section 2 but in each case the type and scope of the reuse is different.

Use Case Model. Mainstream requirements engineering adopts the use case model for grouping, managing and specifying requirements. Not only does this help to manage the complexity of large numbers of requirements but it also presents requirements in a manner that suggests reuse. Well structured use case diagrams support the early identification of possibilities for reuse and guide the designer in the development of software architectures that support reuse. Reuse supported here is at the component level.

Business Model. Patterns that emerge from a comprehensive business model can identify places within an organisation where processes exhibiting similar business requirements might reuse software at the system level. This is fundamentally different from the type of reuse suggested by a use case model. If the business model predates the software system development then it acts as a set of wider business requirements that might be considered when developing software. If the software is purchased as COTS or if its development predates an accurate business model then the business model acts as a guide to the reuse of the software within the business. In either case, reuse is supported.

Agent Model. Terms associated with the definition of agency (autonomous, reactive, intelligent, dynamic environment, uncertainty) capture an implicit expectation that an agent will be reusable. Managing agent reusability requires descriptions of its behaviour that support design. Three forms of reuse are possible in this context: the reuse of an agent within an environment but across different scenarios; reuse of the agent across software environments; and reuse of subcomponents of an agents across agents.

Domain Model. A domain model supports reuse of knowledge at the domain level across projects.

Although a human can certainly satisfy the definition of being an agent, a distinction is made between modelling agents as computational entities in the agent model and humans in the business model for two reasons. Firstly, in domains like military simulation, the agents are computational models of real humans (e.g. fighter pilots) whose behaviour is defined by the domain model (e.g. air combat). The business model is also about the behaviour and activities of humans, but of a different group - the group of humans using the software being developed (e.g. military operational analysts).

4.1 Transitions between Models

The real scope for reuse is seen in the transitions between the models. In practice, there are inevitable iterations through each of these models but there is a logical sequence to consider them. The business model sets the context for the software development and identifies the business activities that will be supported by the software. As these business requirements flow down into the use case models, there are links created between the two models. At the site of these links, it is

possible to identify the "purpose" for which a software requirement exists by tracing back to the business requirement that spawned it.

In precisely the same manner the software requirements expressed in the use case model flow down to the behavioural requirements of the agent that are captured in the agent model. The link between the use case model and the agent model traces the purpose for which the agent behaviour is developed.

The agent behaviour is more complex in that it is also influenced by the specification of the knowledge in the domain model. The use case model provides the reasons for the existence of a particular behaviour in the agent model. The domain model provides the context and broader knowledge that encompasses the agent model. In the links between the domain model and the agent model can be represented the assumptions, constraints and simplifications that help to specify the agent behaviours. The agent model needs to be understood in terms of the software use case model and the domain model. This is because the agents are not built in isolation. Firstly, they must provide a service to the users of the system. Secondly, the behavioural model provides a context for the definition of agent behaviour. Both the use case model and the domain model provide information which can assist in the definition of the agent model.

For example, the business model of a commercial enterprise will allow a business analyst to identify which business processes need to be automated by software systems. This provides the links to the resulting use case models that may be produced to capture the requirements of the software in the context of the business model. A more detailed case study is documented later in this paper.

4.2 Representations of Purpose

Specifying the requirements for an information system is critically tied to an understanding of the manner in which the software will be used – answering *the how question*. This paper will argue that a less intuitive, but stronger, link exists between the *purpose* for which a system is developed and the manner in which the requirements should be specified – *the why question*. Adopting an agent oriented approach to modelling, particularly one that supports purposive descriptions, allows views of a system that support and even encourage reuse.

Purposive descriptions have been associated with each of the four models by other researchers. Purposive descriptions associated with models of agency are not novel [11] and have proven their benefit across a range of fielded applications in subtly different aspects of software engineering. Often models of purpose, intentionality or goal directedness provide the key ingredients to a theory of agency that forms the basis for a language. Related to this are endeavours to augment software requirements specification with features of agency. This clearly supports the specification of systems constructed from purposive languages [9] and promises to have wider application.

Purposive descriptions in use case and requirements modelling emerge in the literature associated with goal directed analysis and design. There is a broad

and deep consideration of goal-directedness in software specification for many different and valid reasons.

> *"The advantage of a goal directed approach is that both the system and its environment can be modelled at a high level of abstraction."* – Haywood and Dart [2]

But the importance of goal-directed requirements modelling has been identified as offering significant advantages [1] and these would extend into reuse. In the world of business modelling the situation is less clear. All manner of approaches to modelling goals in business have been proposed and many of these are based on the UML [14]. In an enterprise wide sense Schonthaler and others define project results and hence software requirements in terms of resulting business performance:

> *"project results are defined in terms of business performance instead of functionality and technical performance."* – Frank Schonthaler Keynote Address Requirements Engineering '02

This links the evaluation of software requirements directly to the success in meeting business goals. Domain modelling, and the area that it shares with business modelling, has also been subject to a variety of approaches based on purposive descriptions. The most comprehensive of these is cognitive systems engineering [12] but others like SPEDE [8] are examples.

In a dynamic information driven world the means by which a goal is achieved is likely to be more dynamic than the existence of the goal itself. As new technologies emerge, opportunities for obtaining competitive advantage hinge on the capacity to fit that technology within the constraints of the business. A view of the business grounded in purposive statements rather than a process-focussed view allows for smoother technology uptake and informs the matching of technology to business requirements. Capturing purpose in the context of requirements specification is facilitated by the adoption of an views of the system that allow for explicit representation of purpose.

5 An Agent Oriented Information System Case Study

This section describes a software system that illustrates the application of the models presented in Section 2. The Royal Australian Air Force's Maritime Patrol Group operate the AP-3C aircraft (see Figure 3[1]). An upgrade of the sensors, particularly the radar, required a rethink of the standard operating procedures that govern the employment of the aircraft. In its role as *advice provider* to the Air Force, the Air Operations Analysis Branch of the Defence Science and Technology Organisation developed a pair of simulation systems to assist with the

[1] See `http://www.cs.mu.oz.au/~michp/aois2003/` for colour versions of figures.

Fig. 3. An upgrade to the sensors on the AP-3C operated by the Royal Australian Air Force necessitated the development of two simulation systems to support the air crews with the development of new standard operating procedures.

analysis and development of tactics. The first of the systems was a constructive[2] simulation used by analysts to examine the impact of modifications to tactics in a large number of scenarios. The second was a *human-in-the-loop* variant that allowed an entire crew to fly the aircraft through many simulated scenarios.

In Figure 1 a part of the business model of Air Operations Analysis (AOA) Branch is shown. This indicates the important roles that are filled by people within the branch and was used to guide the simulation development. This is a standard business model described with the UML and there is nothing innovative about its use in this instance.

The business model is used to provide the context, vision, and motivation for the use case model and the software requirements specification that follows. In Figure 4, only a part of the use case model is shown. This use case diagram is an extension of the normal application of use case modelling to start to account for the presence of agents in the simulator [5]. The standard use case actor in this diagram is the AOA analyst. The actors that exist within the boundaries of the system are a representation of the agents that are required in the simulation. This system is the constructive simulation and consequently the only interaction with the system is with the analyst who runs it.

A significant advantage of adopting a similar notation for representing agents and humans is demonstrated in Figure 5. The specification of the second human-in-the-loop system was largely the same. The difference is that an agent is replaced by a human. In Figure 5, the radar operator agent is replaced by a real radar operator but the rest of the specification can remain unchanged. Of course, the actual implementation of the simulator required a graphical user interface

[2] The simulation community refers to systems that have no human interaction as *constructive*. Typically these simulations are used to explore large parameter spaces and develop statistical conclusions by adopting standard Monte Carlo techniques.

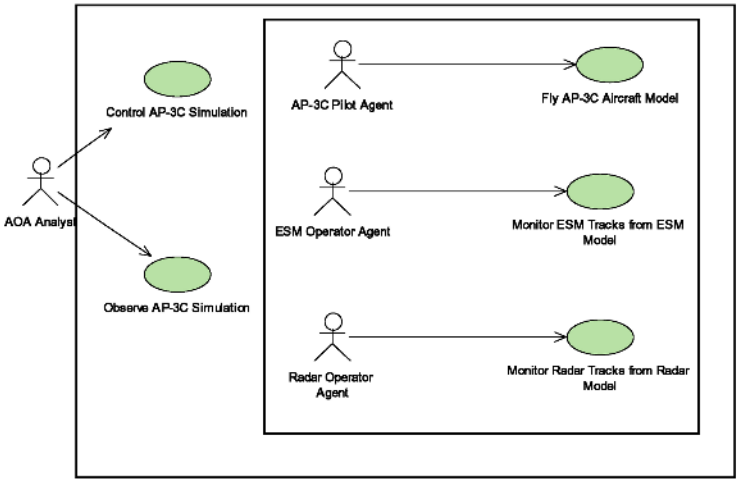

Fig. 4. A use case diagram for the representing behaviours in an AP-3C simulator, for both an analyst and agents in the virtual environment. The outer box represents the simulation software system. The inner box represents the virtual environment. This diagram extends the normal UML representation of actors to represent agents within the simulation system.

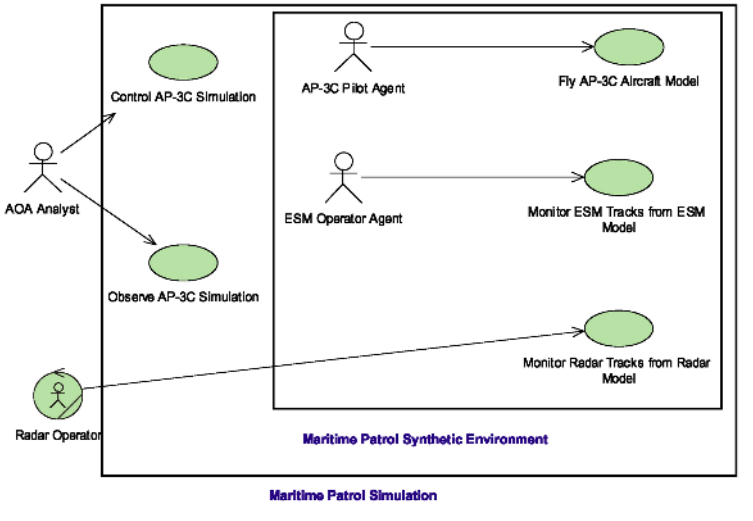

Fig. 5. The use case model of the human-in-the-loop system. The radar operator agent is replaced by real radar operator who uses the system to explore the performance of the simulated aircraft.

for the human but, with the specification as a guide and careful design choices, most of the system was reused in an unmodified form.

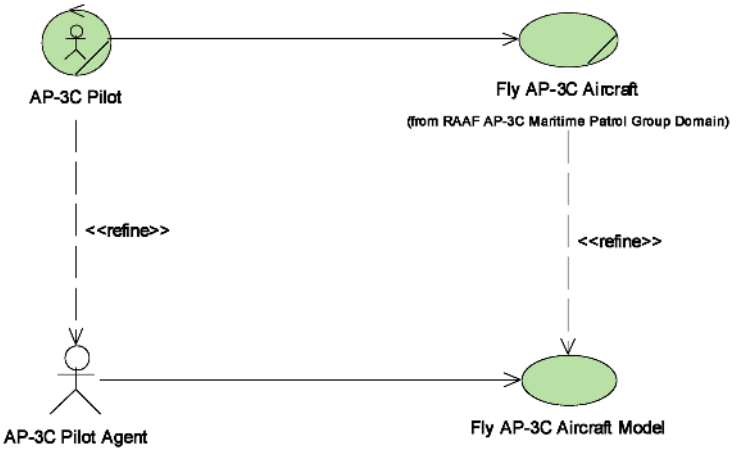

Fig. 6. A diagram that shows the *refine* relationship link between the agents as they are specified and the domain model that stores the knowledge upon which the behaviours of those agents is based.

The tactics, knowledge, procedures, activities and other aspects of the "business" of the Maritime Patrol Group are also modelled. This allows a knowledge base to be built up that provides that information necessary for specifying the agents that simulate the air-crews. Figure 2 shows a part of this domain model indicating the team members and team structure.

When constructing the agent models it is inevitably the case that the particular software requirements dictate that some simplified subset of the domain model will be implemented. Agents are constructed not as complete and accurate simulations of crew-members but as simplified partial representations suitable for the purposes for which the simulation is required. It is important for operations researchers to be able to document and describe the simplifying assumptions that have been made in the development of these agents. These can be reflected in a diagram that shows the refine relationships between the business concepts and their agent counterparts (Figure 6).

Further details of this application have been described in [3] as well as a description of the some of the design advantages that are afforded by the modelling choices presented in this paper [4]. The success of this particular system led to the reuse of much of the system in a second application. The application, known as the Armchair Warrior, is shown in Figure 7.

6 Conclusions

This paper presented four views on requirements modelling. These views are complementary and address different issues and needs that must be considered in the engineering of agent oriented information systems in a business environment. These four models share a common notation as they are all derivatives

Fig. 7. This photograph shows an "Armchair Warrior" exercise in which AP-3C military operators can interact with virtual agents in an AP-3C tactical simulation of an AP-3C maritime patrol mission.

of standard UML use case modelling. This paper presents a unified coherent approach to requirements modelling at these levels that promotes reuse through the explicit capturing of purposive descriptions of the system. Several illustrative examples were provided as was a more detailed real-life case-study. Work is currently under way to address in more detail appropriate techniques for capturing and representing purpose.

The four views on modelling requirements for an information systems were presented at a very high level together with some examples. This was a deliberate attempt to provoke discussion as to what types of views and models are needed to represent requirements adequately for the modern agent oriented information system. Future work will elaborate the details of each of the models using a complete example, showing in a step by step manner how to construct each model and how to make the transition from one requirements model to another.

References

1. A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.
2. E. Haywood and P. Dart. Analysis of software system requirements models. In *Proceedings of Australian Software Engineering Conference*, pages 131–138. IEEE Computer Society Press, July 1996.

3. C. Heinze, M. Cross, S. Goss, T. Josefsson, I. Lloyd, G. Murray, M. Papasimeon, and M. Turner. Agents of change: The impact of intelligent agent technology on the analysis of air operations. In L. Jain, N. Ichalkaranje, and G. Tonfoni, editors, *Advances in Intelligent Systems for Defence*, volume 2 of *Series on Innovative Intelligence*, chapter 6, pages 229–264. World Scientific, River Edge, New Jersey, USA, 1 edition, December 2002.
4. C. Heinze, S. Goss, T. Josefsson, K. Bennett, S. Waugh, I. Lloyd, G. Murray, and J. Oldfield. Interchanging agents and humans in military simulation. *AI Magazine*, 23(2):37–47, Summer 2002. An earlier version of this paper appeared in the Innovative Applications of AI conference, Seattle, 2001.
5. C. Heinze, M. Papasimeon, and S. Goss. Specifying agent behaviour with use cases. In *Proceedings of Pacific Rim Workshop on Multi-Agents, PRIMA*, pages 128–142, 2000.
6. C. Heinze and L. Sterling. Knowledge representation. In *AAMAS Poster Paper 2002*, Bologna, Italy, 2002.
7. P. Kruchten. The rational unified process, an introduction, 2000.
8. N. R. Milton, N. R. Shadbolt, H. D. Cottam, and M. Hamersley. Towards a knowledge technology for knowledge management. *International Journal of Human Computer Studides*, 51(3):615–641, 1999.
9. M. Papasimeon and C. Heinze. Extending the UML for designing jack agents. In *Proceedings of the Australian Software Engineering Conference (ASWEC 01), (to appear)*, Canberra, Australia, 2001.
10. M. Penker and H. Eriksson. *Business Modelling with UML: Business Patterns at Work*. John Wiley and Sons, 2000.
11. A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.
12. J. Rasmussen, A. Pejtersen, and L. Goodstein. *Cognitive Systems Engineering*. Wiley Interscience, 1994.
13. D. Rosenberg and K. Scott. *Use Case Driven Object Modeling with UML*. Addison Wesley, 1999.
14. A. Wegmann and G. Genilloud. The roles of "roles" in use case diagrams. In A. Evans, S. Kent, and B. Selic, editors, *UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings*, volume 1939, pages 210–224. Springer, 2000.

Towards a More Expressive and Refinable Multiagent System Engineering Methodology

Shiva Vafadar, Ahmad Abdollahzadeh Barfouroush, and Mohammad Reza Ayatollahzadeh Shirazi

Intelligent Systems Laboratory
Computer Engineering Faculty
Amirkabir University of Technology
Tehran- Iran
vafadar@aut.ac.ir
{ahmad,ashirazi}@ce.aut.ac.ir

Abstract. In this paper, we improve and extend the MaSE methodology to bridge the gaps in this methodology. First, we propose a methodology improvement process and, based on this process, we report the discovered gaps and weaknesses in the methodology. For removing the reported weaknesses, we introduce the "Role Schema" to document roles properties and the "Knowledge Modeling" step in order to model knowledge of each single agent in the analysis phase of the methodology. We also propose the "Agent–Object model" to decrease design and implementation complexity and improve efficiency of the developed agent-based system. In the improvement process, for evaluating the proposed refinements and extensions we have analyzed and designed the CASBA multiagent system with the improved MaSE. We will show that these improvements will increase expressiveness and refinability of the methodology and maintainability of the developed agent-based system.

1 Introduction

Now agent-based solutions are used in some industrial, commercial, medical, networking and educational application domains. These software solutions are mainly complex, open and distributed. Agent-oriented software engineering is a powerful way of approaching large-scale software engineering problems and developing agent-based systems [6]. We consider agent-oriented software engineering as a layered technology that encompasses agent-oriented software process models, agent-based system architectures, agent-oriented methods and agent development tools [2]. Also, there are activities such as software measurement, configuration management and software quality assurance for agent-based system for complementing the main tasks and steps in agent-oriented software engineering [2]. There are some agent-oriented methods for analysis and design of agent systems. An overview of such methods is presented in [5, 16]. The early versions of these methods have many weaknesses in analysis and design of agent-based systems. Therefore, researchers are extending and improving these methods for agent-based systems so that they can better guide developers to develop agent-based systems.

P. Giorgini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 126–141, 2004.

One of the better developed agent-oriented methodologies is Gaia [15]. It has been improved to model complex open systems [7] and to model aspects of Internet applications [17]. The Multiagent System Engineering (MaSE) methodology is another one, which takes an initial system specification and produces a set of formal design documents in a graphically based style [14]. This methodology has been improved by researchers in order to cover and model some aspects of multiagent systems such as mobility [11], ontology [4] and organizational rules [3]. Although MaSE has improved in many aspects by its developers in recent years, it still has some weaknesses that should be discovered and improved.

In this paper, we extend and improve the MaSE methodology to bridge the gaps in the methodology activities and artefacts. We will introduce a schema to document role properties and a "Knowledge Modeling" step in order to model knowledge of each single agent in the analysis phase of the methodology. Also, we will propose an "Agent–Object" model to decrease design and implementation complexity. In our improvement processes, we have used two different case studies for evaluating the methodology and analyzing the existing gaps in the MaSE. Since a methodology should be comprehensible to both experts and novices (accessibility criteria), in the first improvement iteration we have studied the methodology from a novice's point of view.

The remainder of this paper reports our MaSE improvement process and the results that we have obtained in this process. In section 2, we describe our improvement process and the methodology evaluation criteria. In section 3, we review the MaSE methodology and its features. In section 4, we describe in detail the identified weaknesses in the MaSE during development of our first case study. In section 5, we propose E_x-MaSE, the new version of the MaSE methodology and our extensions in terms of improved and new models and steps. In section 6, we will present our analysis and design of an agent-based system by the E_x-MaSE. In section 7, we will present our conclusions and further works.

2 Our Methodology Improvement Process

In order to improve the MaSE methodology, we use a methodology improvement process. Fig. 1 shows the proposed process. According to this process, first we analyze and design an agent-based system by following the MaSE methodology. The first case study is an electronic bookshop (E-shop), which could be implemented as a multiagent system. Then, we evaluate the methodology based on the produced software artefacts in the different phases and the developed multiagent E-Shop application. In our evaluation process, we use evaluation criteria such as preciseness, accessibility, expressiveness and a smooth transition between the development phases [12]. During the analysis and design of our case study, we had some problems in many steps of the methodology and we needed some models to design and implement the agent-based system, which the methodology does not provide.

Then, we try to improve the methodology according to the evaluation results. The improvement process continues its iteration with analysis and design of another case study, this time by using the improved methodology. The process iterates until all agent-based systems abstractions and semantics can be modelled effectively by the methodology and the methodology satisfies all the desirable criteria. Using different

case studies at different levels of complexities and in a variety of application domains can help us to discover the gaps and weaknesses in the methodology. In the next sections, we report the results and our contributions that we have reached them by following the improvement process.

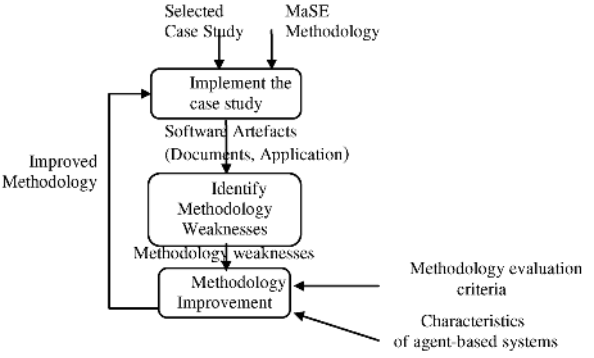

Fig. 1. Our Improvement Process.

3 The MaSE Methodology

Today, there are several agent-oriented methodologies [5,16]. The majority of them are based on existing object oriented or knowledge-based methodologies and the syntax of many of the models were taken from the Unified Modeling Language. The Multiagent System Engineering (MaSE) is one of the methodologies that has tried to be independent of a particular multiagent system architecture, agent architecture, programming language or message-passing system [14]. It also provides a tool, called agentTool, which supports different phases of the methodology. Our study shows that, in comparison with some of the other methodologies, it can better model agent properties such as autonomy, proactivness and goal-directed behavior. Also, it provides more expressive models for modelling communication and conversion in multiagent systems. It also meets some of the software engineering criteria such as preciseness, accessibility, refinability and maintainability [1].

The MaSE methodology takes an initial system specification and produces a set of formal design documents in a graphically based style. There are two basic phases in the MaSE: analysis and design. The first phase, Analysis, includes three steps: Capturing Goals, Applying Use Cases and Refining Roles. The first step, Capturing Goals, takes user requirements and turns them to top-level system goals. Using system-level use cases and defining sequence charts in the Applying Use Cases step, an initial set of system roles and communications paths are defined. In the Refining Roles step, using the system goals and roles identified in the use cases, an initial set of roles is refined and tasks to accomplish each goal are defined. The design phase has four steps: Creating Agent Classes, Constructing Conversations, Assembling Agent Classes and System Design. In the first step, Creating Agent Classes, specific agent classes are defined to fill the roles defined in the analysis phase. Then, after determining the number and types of agent classes in the system, conversations between those agent classes are constructed and the internal components that comprise the agent

classes are defined. At the end, the designer defines the number of individual agents, their locations and other system specific items.

MaSE has been successfully applied in numerous graduate-level projects as well as several research projects [9,14]. Also, this methodology has been improved by researchers in order to cover and model some aspects of multiagent systems such as mobility [11], ontology [4] and organizational rules [3]. Therefore, according to our improvement process, the MaSE methodology is improving and maturing but it still has some weak points. In the following section, we describe the weaknesses of the MaSE discovered in our research.

4 MaSE Weaknesses

We evaluated the MaSE methodology according to our proposed improvement process. The first case study that we used for evaluation of the MaSE methodology was an Electronic bookshop (E-Shop). Our first case study must be simple enough to help a novice developer to understand the methodology and to evaluate how the methodology helps designer through the analysis and design phases and how much it provides the necessary information for a programmer during the implementation phase. During this evaluation, our focus was on the implementation requirements that the methodology does not address in detail. A smooth transition from the analysis to the design phase and among different analysis and design steps has been considered as well. We also considered some criteria in our evaluation process such as expressiveness and refinability of the methodology [12], as well as maintainability and efficiency of the developed system by considering necessary activities in the methodology. In the first iteration of our improvement process, we identified the following weaknesses during analysis, design and implementation of the case study.

1. **Existing gap between analysis and design phases:** One of the most important weaknesses of the MaSE methodology is the gap existing between analysis and design phases. For example, the "Assembling Agent Classes" step in which agent architecture is constructed is an isolated step that does not have appropriate connection with other analysis and design activities. In this step, the designer needs information that is not provided in the earlier steps. Therefore, he/she should go back to the first step and analyse the role requirements to gather necessary information for decision making about the appropriate agent architecture.
2. **Lack of knowledge modelling in the methodology:** Knowledge is an essential component of an agent. An agent has a knowledge component and a mechanism for operating on it or drawing inferences from its knowledge. In the MaSE, the knowledge of an agent is represented at the agent level design in the "Assembling Agent Classes" step. According to the agent's knowledge, the designer selects the agent architecture and its components. Since the MaSE is a communication-oriented methodology, its focus is on the interactions between agents. Since the previous extension to the MaSE for ontology modelling [4] focuses on the data model of the system and not on the mechanism for operating on or drawing inferences from it, modelling the internal knowledge of agents such as rules or plans is not addressed in the methodology.

3. **Mapping all roles to agent abstraction:** In the "Creating Agent Classes" design step, all refined roles are mapped to the corresponding agents. Although some of them are independent roles, they do not have the desired properties of agents such as autonomy and reactiveness. We believe that they are just objects in a multiagent system and in such cases we should model such entities as objects.
4. **Weak documentation:** In the MaSE, documentation of many aspects of an agent-based system is implicit. For example, roles are just documented by the "Role Diagram" and their tasks in the analysis phase. Since a good documentation is needed for maintenance of a system, the methodology should guide the software engineer to produce the necessary documents and artefacts for better maintenance of the developed system. Explicit documentation for roles helps the methodology to achieve this goal.
5. **Lack of clear basic definitions and guidelines**: MaSE does not have an explicit definition for goal, role, task and activity. For example, the overlap between the definitions for goals and tasks causes conflict between these two concepts. Consequently, the analyst should go back to the first analysis step and refine the analyzed goals. These uncertainties in definitions increase the time and cost of performing analysis activities.
6. **Problem in modelling interactions:** MaSE uses UML sequence diagram for modelling role interactions so it cannot model some specific characteristics of agents such as concurrent threads of interactions among roles. We believe that, by using AUML extensions [10], we can decrease the number of necessary sequence diagrams.

In our improved methodology, we have proposed some models and steps to remove some of the above weaknesses in the MaSE. The proposed improvements can lead us to a more expressive and refinable methodology and more efficient and maintainable agent-based systems. We will describe our improvements in the following section.

5 Proposed Improvements and Extensions for the MaSE

To improve and extend the MaSE methodology, we have added the "Role Schema" to the “Refining Roles” step and the "Knowledge Modelling" step in the analysis phase and we have extended the "Agent Modelling" in "Creating Agent Classes" step to "Agent-Object Modelling". We call this new version of the methodology, E_x-MaSE. Fig. 2 shows the E_x-MaSE steps. In this figure, white boxes show the steps of the original methodology and grey boxes show the extended steps and models. In the following subsections, we will describe improvements and/or extensions in detail.

5.1 Role Schema

For explicit documentation of roles, we have introduced the "Role Schema" as a documentation tool to the "Refining Roles" step in the MaSE analysis phase. In this

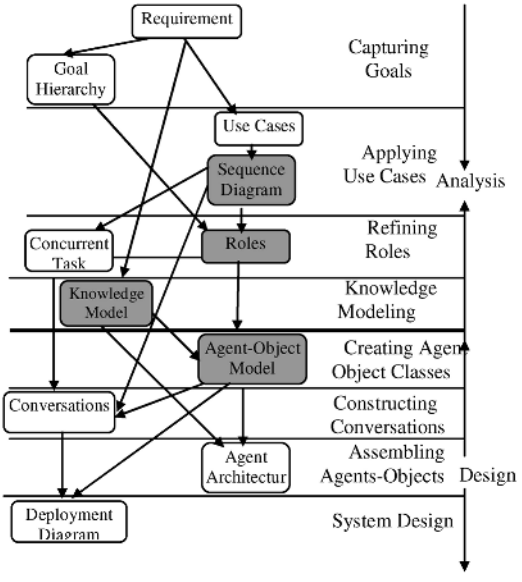

Fig. 2. E_x- MaSE Steps and Models.

extension, we added a Role Schema for each role in the system. Fig. 3 illustrates a "Role Schema" that draws together the various attributes of a role. This information is typically quite informal and therefore free text is preferred to a graphical notation. External and physical resources slots are needed for I-O interface components of all the predefined architectures of the agents in the design phase. Permissions define data objects that are accessible by this role.

Role Schema: Name of role

Description:
Short English Description of role

Goals:
Goals associated with the role

External Resources:
External data sources ranging from HTML files to databases that agent need them

Physical Resources:
Sensors and effectors related to agent

Permissions:
Rights associate with the role to access data objects

Critical situations:
Safety conditions that cause to avoid failure of the role

Fig. 3. Role Schema.

A refinable methodology should provide a clear path for refining a model through a gradual stage from analysis to implementation [12]. In the original MaSE, a designer needs some information for the I-O interface component of internal agent architecture that is not gathered in previous steps. By using these schemata, we gather required information in analysis phase in order to choose predefined components of agent architecture in the design phase. This documentation will make transition from

the analysis phase to design phase smoother. Therefore, by using these schemata, the methodology will become more refinable.

Also by explicit documentation of role requirements, the scope of each role is defined. Since documentation is one of the tools that helps the methodology to produce more maintainable systems, adding "Role Schemata" makes the developed system more maintainable.

5.2 Knowledge Model

We have proposed a “Knowledge Model” for modelling knowledge of the roles in the analysis phase. A knowledge component is necessary for decision-making about the appropriate agent architecture and should be provided in the analysis phase. By using "Use Case Model" and "Role Schemata", the system analyst can identify the required knowledge for each role to achieve its associated goals. The architectural styles that are used in the "Assembling Agent Classes" step have some common knowledge components that can be categorized as rules, facts and plans. The rules are used to interpret the stimulus and generate a response if appropriate. The analyst should model each rule according to the schema that is shown in Fig. 4.

Rule Schema: Rule Name

Role(s) Name:
List of roles which act base on this rule
Description:
An English description of the rule.
Preconditions:
Precondition that should considered
Post conditions:
Behavior of the role base on the rule.

Fig. 4. RuleSchema.

In general, a plan consists of one or more steps, each of which contains a number of attributes. Also, for each predefined plan used by a role, we propose a “Plan Schema”, shown in Fig. 5. This schema can be used for documenting dynamic and static plans of a role. For each fact in the system, the analyst should specify a name and a resource that provides the fact and update conditions. The facts contain any predefined information the agent may need as well as any new information derived by the agent. It should be mentioned that it is not necessary to represent any type of knowledge for each role. In some cases, a role may not use any special knowledge. By adding this step to the analysis phase, we model knowledge level aspects of the system in the analysis phase. An expressive agent-oriented method should represent some aspects of an agent-based system such as the structure of the system, the knowledge encapsulation within the system, resource constraints etc. [12]. By adding a "Knowledge Modelling" step to the analysis phase, the methodology will be able to model and document the required knowledge of each role to achieve its goals. Therefore, by modelling the knowledge encapsulation within the agent-based system, the expressiveness of the methodology is increased.

A good agent-oriented methodology should be refinable. This means that it should provide a clear path for refining a model through gradual stages to reach an implementation level [12]. The proposed "Knowledge Model" increases the refinability of

the MaSE by adding a knowledge modelling activity to the analysis phase of the methodology. By using this model, the designer can easily decide about the appropriate agent architecture of the system based on the identified rules, plans and facts of the system. For example, if the analyst defines some predefined plans for a role, the designer can decide that the related agent should use a planning architectural style to achieve its goals. Alternatively, in the knowledge modelling step, if a role just uses some rules for reacting to its environment, the designer will easily choose a reactive architecture as the internal agent architecture.

Plan Schema: Plan Name

Role(s) Name:
List of roles which act base on this plan
Description:
An English description of the plan.
Type:
Dynamic or Static
Priority:
Priority of the plan in compare with other plans
States:

State Number	Precondition	Post condition
State number of plan	Preconditions of State	Post conditions of state

Fig. 5. Plan Schema.

5.3 Agent-Object Model

A sound principle in system development states that one should always attempt to develop the simplest solution for any given problem. Also, a good agent-oriented methodology should encourage developers to achieve the reasonable decomposition of entities. In the MaSE, all refined roles in the analysis phase are mapped to the corresponding agent classes in the design phase. In this way, every entity in a system is modelled as an agent. However, there are some roles in the systems that are simpler than an agent. A role that has a deterministic response to messages and does not have the desired characteristics of an agent is essentially a simple object in an agent-based system. Because objects are simpler computational entities than software agents, in such cases it is more efficient to use object abstraction. If one follows this guideline in design of an agent-oriented analysis and design method, the resulting method can help the system analyst to choose whether individual computational entities in the system should be modelled and implemented based on either software agents or objects. Thus, we have refined "Agent Model" to "Agent-Object Model". In this model, the designer can map a role to either an agent or an object. If a role does not use any rule and fact or does not act based on any predefined plan, which has been provided in the proposed "Knowledge Modelling" step, it is not an agent and can be modelled as an object.

Since objects are simpler computational entities than agents to design and implement, by adequate decomposition of roles to agents or objects, we can decrease the design and implementation time of the system and the complexity of the developed system as well. By using an "Agent–Object" model, the designer can consider the resource constraints of the system. Objects need less memory and CPU resources at

run time. Therefore, the designer can map each role to an agent or an object based on the role characteristics and system resources. Considering system resources is one of the characteristics of an expressive methodology [12]. Therefore “Agent–Object" model increases the expressiveness of MaSE methodology.

6 Analyses and Design of the CASBA with the E_x-MaSE

In the second iteration of our improvement process, we used the CASBA [13] as the second case study for evaluating our improvements and refinements in the MaSE methodology. The aim of the CASBA project is developing an electronic marketplace using intelligent agent technology. The CASBA system offers automated negotiation with wide range of transactions such as brokerage and a variety of agent-driven automated auction types. All participants in the automated auctions are agents that can take one of four roles. The administrator agent manages the whole market and the auctioneer agents handle a single auction. The buy and sell agents execute the orders of their users. The CASBA supports English, Dutch, First price sealed bid and Vickery auctions.

We chose the CASBA project because we needed a more complex system than E-shop multiagent system. In this way, we continued in the E-commerce field and selected an agent-based market supporting multi-auction protocols. The multi-auction property of CASBA makes it more complex. Also, we can develop it incrementally by implementing each auction type in a separate iteration. This property can help us to evaluate the extendibility and maintainability of the developed system in different iterations of development. In the following, we analysis and design the CASBA system by the E_x-MaSE.

Capturing Goals

The first phase in E_x-MaSE is “Capturing Goals” which takes system requirements and constructs a " Goal Hierarchy Diagram". In the CASBA system we have a "Goal Hierarchy Diagram" as shown in Fig. 6.

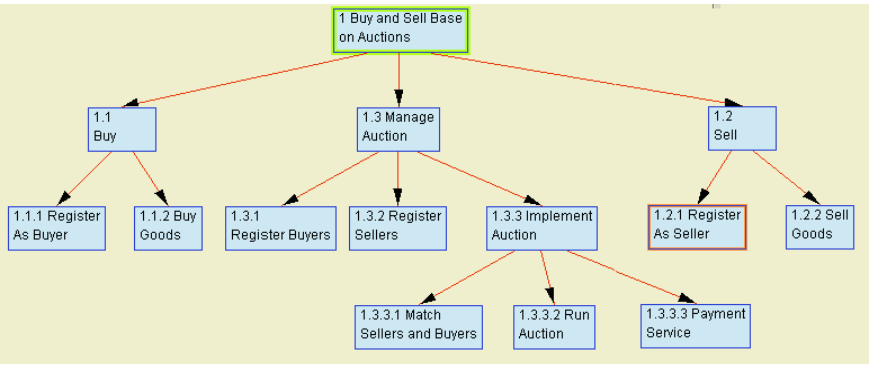

Fig. 6. Goal Hierarchy Diagram of the CASBA System.

Applying Use Cases

In this phase, the goals identified in the previous phase are translated to the roles. Use cases are constructed based on system requirements and communication between roles in the system. Then, use cases are reconstructed and instantiated as "Sequence Diagrams". Fig. 7 shows the run auction "Sequence Diagram" for the CASBA system.

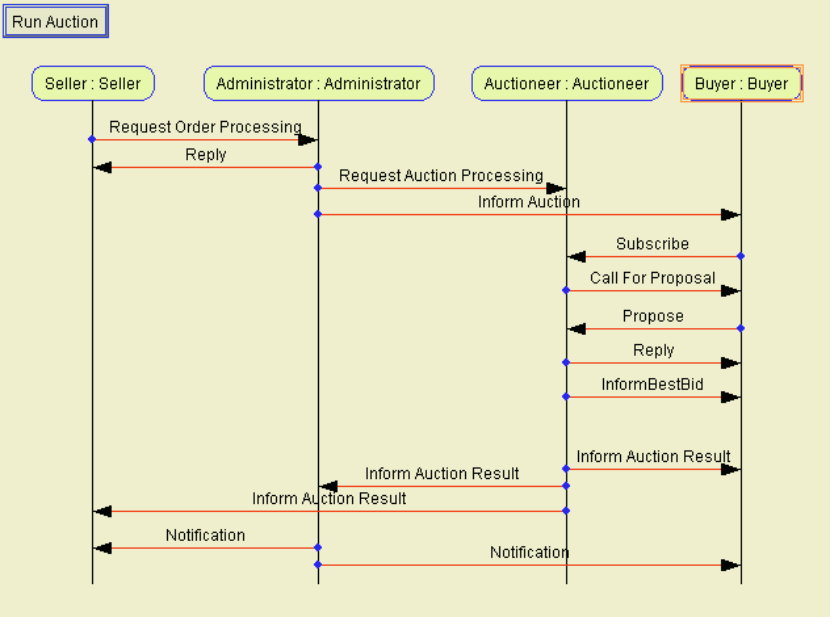

Fig. 7. Run Auction Sequence Diagram of the CASBA System.

Refining Roles

In this step, identified roles are reviewed to ensure that all the necessary roles are considered. Then, different roles are combined or decomposed based on responsibilities and interaction among them. We have proposed a "Role Schema" for documenting each role in the system as an artefact of this step. The following schema shows the "Administrator Role Schema" of the CASBA system.

Role Schema: Administrator
Description: Admin is responsible to receive all requests from sellers and buyers and introduce matched seller and buyers to an auctioneer, which is responsible for that auction. Also it manages all the auctions in the system.
Goals: Manage Auctions (1.3)
External Resources: None
Physical Resources: time
Permissions: Sellers information (read/write), Buyers information (read/write), auction goods (read/write), Auction rules (read), Active auctioneers (read/write), Auction results (read/write),
Critical situation: Invalid auction type orders, payment problems

For each role, related tasks are identified. Fig. 8 shows "Role Model" of the CASBA. After identifying roles, their tasks are explained by "State Diagrams". Separate concurrent state diagrams show how a role can achieve its associated goals including how it interacts with other roles. Fig. 9 demonstrates the "Inform Auction Task State Diagram" of the manager role.

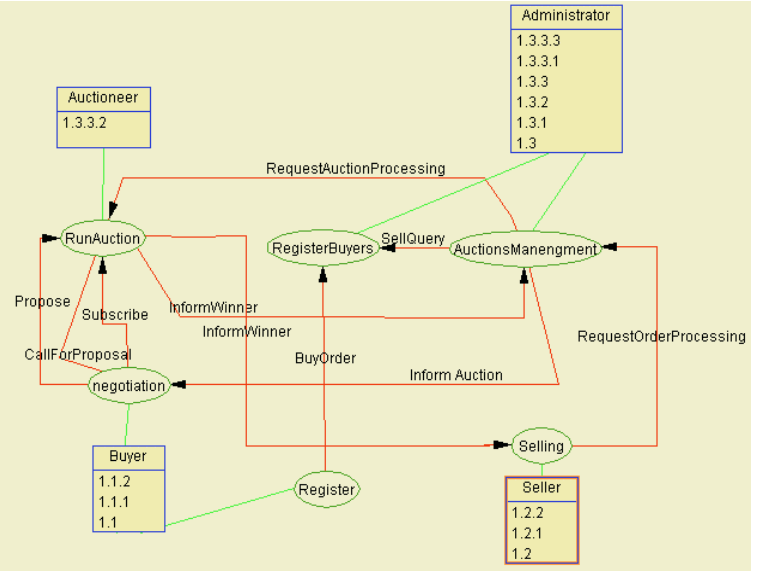

Fig. 8. Role Model of the CASBA System.

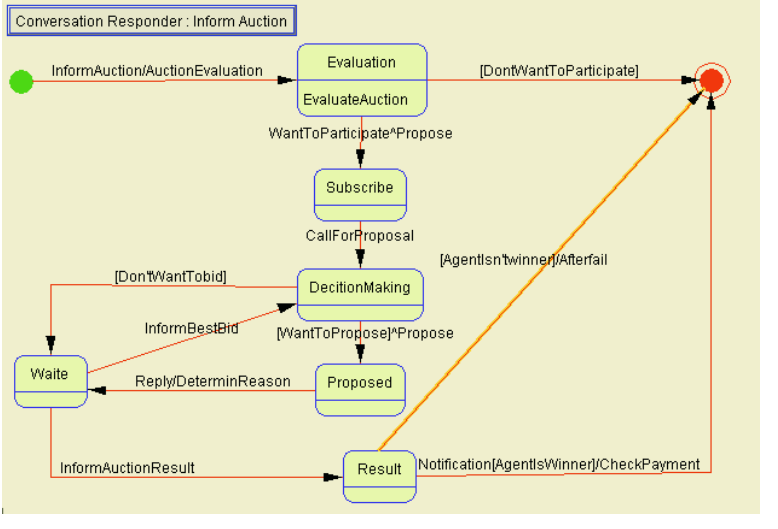

Fig. 9. Inform Auction Task Diagram of the CASBA System.

Knowledge Modelling

We added a "Knowledge Modelling" step in order to analyse the required knowledge of each role to achieve its goals. By using "Sequence Diagram" and "Role Schemata" rules, plans and facts of each role are identified by the analyst. The following schemata show a rule of English auction, which is used by an auctioneer in evaluating the received bids, and a rule that is used by administrator in starting auctions.

Rule Schema: English Auction bidding
Role(s) Name: Auctioneer
Description: In English auction, bids less than best bid (or initial bid at first) and bids without price, should reject.
Preconditions: Newbid < Bestbid ‖ newbid=null
Post conditions: Reject bidding

Rule Schema: Start Auction
Role(s) Name: Administrator,
Description: If matched buyers in an auction are less than a predefined threshold, auction cannot start and seller should wait.
Preconditions: Matched Buyers< Auction threshold
Post conditions: add seller to waiting list / check seller in each new request

Also, each role needs a set of facts of the system. These facts can be provided by user or other roles. The following table is an instance of facts that the Administrator role uses in the CASBA system. Using update conditions, the analyst can check related rules and plans and complete the rules, plans and facts iteratively**.**

Role Name: Administrator			
Fact name	**Resource**	**Related service**	**Update condition**
Initial price of Auction	Seller	Send to auctioneer	None
Seller waiting list	Administrator	Matching sellers and buyers	Seller with inadequate buyers (add) Requesting by a buyer for a waiting seller good (remove)

Creating Agent-Object Classes

In the original MaSE in the "Creating Agent Classes" step, agents are documented in an "Agent Class Diagram" according to the identified roles in the analysis phase. Communications between agents are mapped according to the identified communications between roles in the analysis phase. We have refined this step as "Creating Agent-Object Classes". In this step, roles can be mapped to either objects or agents. Fig. 10 shows the "Agent-Object Class Diagram" of the CASBA. In this system. seller is not an agent because in our proposed "Knowledge Modelling" step it does not use any rules or plans. It just receives a request from the user and sends it to the administrator of the system. Therefore, we model it as an object in a multiagent system. In the "Agent-Object Diagram", an agent is represented by a cube and an object is represented by a rectangle. In this diagram, relationships among agents and objects represent a communication relationship between these entities.

Constructing Conversations

In MaSE, a conversation is a coordinator protocol between two agents and it documents a communication class diagram for the initiator and responder of each conversation by finite state machines. Since, in our improvement, the designed entities can

be both agents and objects, in this step the designer should construct conversation(s) for complex objects. Fig. 11 shows the "Inform Auction state diagram" for the buyer agent as responder.

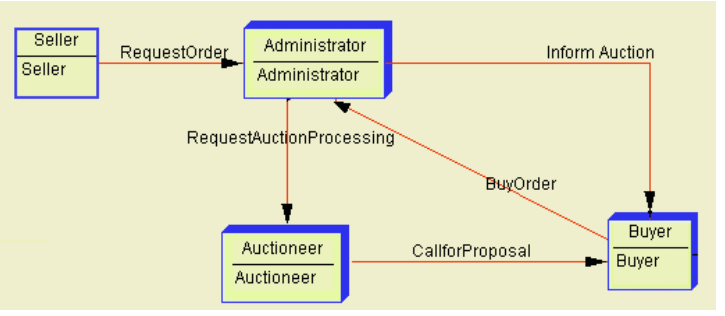

Fig. 10. Agent- object Class Diagram of the CASBA System.

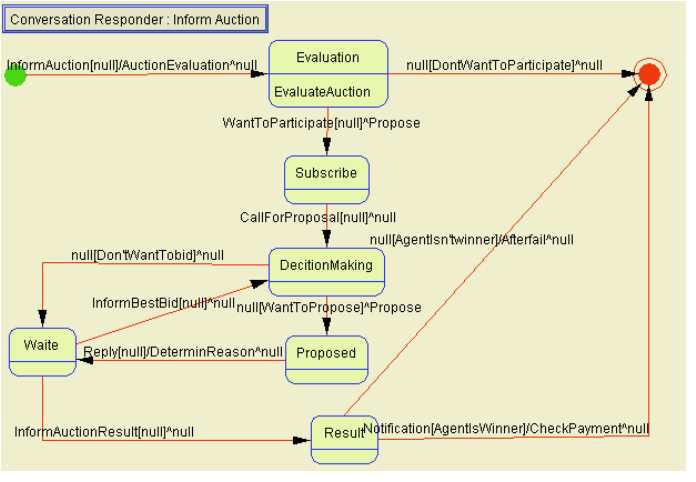

Fig. 11. Inform Auction Conversation (Responder).

Assembling Agent-Object Classes

In this step, internals of agent classes and more detailed objects are created. Five different architecture styles are defined for agents: BDI, reactive, planning, knowledge base and user defined. Each architectural style includes predefined components [10]. By using the proposed “Knowledge Model”, the designer can decide about the agent architecture. In the CASBA system, we have three agents: Administrator, Auctioneer and Buyer. Since all of them just use rules as their knowledge in the "Knowledge Model", we design them as reactive agents. Also in this step, object classes should be constructed. Similar to object-oriented design, this step is concerned with defining attribute types, operations and more detailed associations among objects.

System Design

The final design step is defining instances of agent and object classes in the system. It uses a "Deployment Diagram" to document numbers, types and location of each class. Fig. 12 illustrates the "Deployment Diagram" of the CASBA system.

For consistency of models in the methodology, we have refined the "Deployment Diagram". In this figure, Seller is an instance of the Seller object class that is shown by a two-layer rectangle, while Administrator, Auctioneer and Buyer are instances of related agent classes in the system, which are shown by cubes.

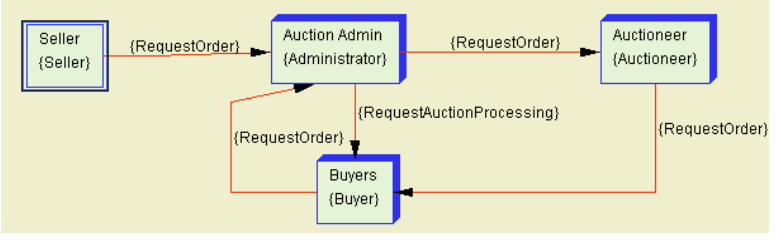

Fig. 12. Deployment Diagram of the CASBA System.

Our proposed extensions and improvements in the MaSE methodology make the transition between different steps of the methodology smoother than in its original version. By performing the proposed activities and producing the necessary artefacts in our second case study, we developed the CASBA system more easily than the first case study.

7 Conclusions and Further Works

In this paper, we have proposed a methodology improvement process and, by following this process, we have shown how the MaSE methodology can be extended in order to bridge the gaps in its analysis and design phases. In the new version of the methodology, E_x-MaSE, we added the “Knowledge Modelling” step into the analysis phase, introduced a Role Schema in the "Refining Role" step and refined the "Agent Model" of the system into an “Agent-Object Model” in the design phase. We discussed how the proposed "Role Schema", “Knowledge Model” and “Agent-Object Model” would improve refinability and expressiveness of the methodology as well as the maintainability and efficiency of the developed system.

In order to evaluate our proposed extensions and improvements, we analyzed and designed the CASBA multiagent system with the improved methodology in another iteration of improvement process. In the analysis and design of the CASBA multiagent system, we did not have any difficulty during the internal agent architecture design in comparison to our first case study. Here, using the proposed “Knowledge Model” in the analysis phase helped us to make effective decisions about the appropriate agent architecture. We also spent less time in designing the system because of the efficient decomposition of the entities into either agents or objects. Also, we believe that this capability in the methodology leads us to a more efficient agent-based system.

We have evaluated E_x-MaSE based on different approaches such as comparison with other methodologies based on most common aspects of multiagent systems [1]. In comparison with some of the other methodologies such as Gaia, Message, MAS-CommonKADS, MaSE and its previous extensions, the E_x-MaSE methodology supports more aspects of the agent-based systems. We have also evaluated it according to software engineering criteria and compared it with MaSE [1]. Our evaluations show that the E_x-MaSE methodology is more expressive, refinable, accessible and extendible than the MaSE methodology.

In the next improvement iteration, we will evaluate the methodology according to the other criteria such as reusability of the methodology artefacts and maintainability of the devloped system. Also in this process, we will work on details of the Agent-Object modelling technique and, especially, the agent-object interaction model. Extending agentTool to support new steps and models is considered as well. Our final goal is to propose a new agent-oriented methodology with all the desired characteristics of a good methodology.

References

1. Abdollahzadeh Barfouroush A., Vafadar, S.: E_x-MaSE: An Extended and Expressive Multiagent system Engineering, Proceedings of the 2003th Arab conference on Information technology conference (ACIT2003), Alexandria, Egypt.
2. Ayatollahzadeh Shirazi, M. R., Abdollahzadeh, A.: Agent-based Software Engineering as a Layered Technology, Proceedings of the Workshop on Agents for Information Management, The First EurAsian Conference on Advances in Information and Communication Technology, Iran (2002)
3. DeLoach, S. A.: Modeling Organizational Rules in the Multiagent Systems Engineering Methodology. Proceedings of the 15th Canadian Conference on Artificial Intelligence. Calgary, Canada (2002)
4. DiLeo, J., Jacobs, T., DeLoach, S.: Integrating Ontologies into Multiagent Systems Engineering, 4th international bi-conference workshop on agent- oriented Information systems (AOIS 2002), Bologna, Italy (2002)
5. Iglesias, C. A., Garijo, M., Gonzales, J.C.: A survey of Agent-Oriented Methodologies, Proceedings of the 5th International Workshop on Intelligent Agents, Agent Theories, Architectures, and Languages (ATAL-98)
6. Jennings, N.: On Agent-based Software Engineering. Artificial Intelligence: 117 (2000) 277-296
7. [7] Juan, T., Pearce A., Sterling L.: ROADMAP: Extending the Gaia Methodology for Complex Open Systems, Proceedings of the first international joint conference on Autonomous agents and multiagent systems (AAMAS2002), July, Bologna,
8. Odell, J., Parunak, V., Bauer B.: Extending UML for Agents, AOIS Workshop at AAAI (2000).
9. O'Malley, S. A., DeLoach, S. A., Determining When to Use an Agent-Oriented Software Engineering Paradigm, Proceedings of the Second International Workshop On Agent-Oriented Software Engineering (AOSE-2001), Montreal, Canada, 2001.
10. Robinson, D.: A Component Based Approach to Agent Specification, Department of electrical and computer engineering Air Force Institute of Technology, M.Sc. Thesis (2000).
11. Self, A., DeLoach, S.: Designing and Specifying Mobility within the Multiagent Systems Engineering Methodology. Proceedings of the Eighteenth Annual ACM Symposium on Applied Computing, Melbourne, Florida, USA (2003)

12. Shehory, O., Sturm, A., Evaluation of Modeling Techniques for Agent- Based Systems, Proceedings of The Fifth International Conference on Autonomous Agents, pp. 624-631, 2001.
13. Vetter, M., Pitsch, S.: An Agent-based Market Supporting Multiple Auction Protocols, workshop on Agents for Electronic Commerce and Managing the Internet-Enabled Supply Chain, Third International Conference on AUTONOMOUS AGENTS, Washington, (1999)
14. DeLoach, S., Wood, M., Sparkman, C., Multiagent, Systems Engineering, The International Journal of Software Engineering and Knowledge Engineering, 11(3): 231-258, 2001.
15. Wooldridge, M., Jennings, N., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems. 3 (3) (2000) 285-312
16. Yu, E., Cysneiros, L.M: Agent-Oriented Methodologies-Towards a Challenge Exemplar. Proceedings of the 4th Int. Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS 2002) Toronto, Canada (2002)
17. Zambonelli, F., Jennings, N., Omicini, A., Wooldridge, M.: Agent-Oriented Software Engineering for Internet Applications, Coordination of Internet Agents: Models, Technologies, and Applications, Chapter 13. Springer-Verlag, (2001).

A Pattern Language for Motivating the Use of Agents

Michael Weiss

School of Computer Science, Carleton University, Ottawa, Canada
weiss@scs.carleton.ca

Abstract. The advantages of the agent-based approach are still not widely recognized outside the agent research community. We believe that there is a need for guidelines that summarize the key benefits of the agent approach to non-agent experts. Agent patterns can provide that guidance. The agent community has already started to use patterns for describing best practices of agent design. However, these patterns tend to pre-suppose that the decision to follow an agent approach has already been made. In this paper we present a pattern language – a set of patterns that build on each other – as a starting point for more specific agent pattern languages. It documents the forces – the drivers, and design trade-offs – involved in agent-based design, and a set of generic patterns that apply to all agent systems. These patterns introduce and motivate the concepts of agent society, roles, common vocabulary, delegation and mediation. The paper illustrates the application of these patterns with a case study on information agents for filtering news.

1 Introduction

Agents are rapidly emerging as a new paradigm for developing software applications. They are being used in an increasing variety of applications, ranging from relatively small systems such as personal assistants to large and open mission-critical systems such as switches, electronic marketplaces or health care information systems.

There is no universally accepted definition of the notion of an *agent*. However, the following four properties are widely accepted as characterizing agents: *autonomy, social ability, reactivity* and *proactiveness* [29]. Agents are autonomous computational entities (autonomy), which interact with their environment (reactivity) and other agents (social ability) in order to achieve their own goals (proactiveness).

Agents typically represent different users, on whose behalf they act. Most interesting agent-based systems are thus collections of collaborating autonomous agents (typically referred to as *multi-agent systems*), each representing an independent locus of control. Multiple agents can, of course, be acting on behalf of the *same* user.

Agents also provide an appropriate *metaphor* for conceptualizing certain applications, as the behaviour of agents more closely reflects that of the users whose work they are delegated to perform or support. This reflects the fact that most complex software systems support the activities of a group of users, not individual users. Such agents are then treated as actors in a system that comprises human actors as well.

P. Giorgini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 142–157, 2004.

The following domain characteristics are commonly quoted as *reasons* for adopting agent technology: an inherent distribution of data, control, knowledge or resources; the system can be naturally regarded as a society of autonomous collaborating entities; and legacy components must be made to interoperate with new applications [24].

However, the advantages of the agent-based approach are still not widely recognized outside the agent research community. While there are several papers discussing the differences between agents and objects [12, 13], on the one side, and agents and components [8], on the other, these papers do not provide actual guidelines for assessing whether a particular development project can benefit from using an agent-based approach. Patterns, on the other hand, are an effective way of guiding non-experts [11].

In this paper, we use patterns as a way of motivating the use of agents. In the following sections, we first summarize the related work on agent patterns and make a case for documenting a set of generic agent patterns that can be used as a conceptual framework and context for documenting more specific agent patterns. We then argue that agent pattern authors should organize their patterns in the form of pattern languages and present a template for agent pattern languages. This is followed by a description of the forces involved in agent-based design and of the patterns of our pattern language that introduce key agent concepts to non-agent experts. We conclude by illustrating how these patterns can be applied with a case study.

2 Related Work

Patterns are reusable solutions to recurring design problems. They provide a vocabulary for communicating these solutions to others. The documentation of a pattern goes beyond documenting a problem and its solution. It also describes the *forces* or design constraints that give rise to the proposed solution [1]. These are the undocumented and generally misunderstood features of a design. Forces can be thought of as pushing or pulling the problem towards different solutions. A good pattern balances the forces.

There is by now a growing literature on the use of patterns to capture common design practices for agent systems [2, 12, 9, 16, 28]. The separate notion of an *agent pattern* can be justified by differences between the way agents and objects communicate, their level of autonomy and social ability [7]. Agent patterns are documented in a similar manner as other software patterns, except for the structure of an agent pattern where we will make use of *role models* [20, 13]. The distinction between role models and collaboration diagrams is the level of abstraction: a collaboration diagram shows the interaction of instances, whereas a role model shows the interaction of roles.

Table 1 provides a summary of the existing work on agent patterns and suggests a classification along two dimensions: domain-dependence (whether the patterns are domain-specific, or domain-independent) and focus (whether the patterns focus on the design of agent internals, the design of agent systems or the design process itself).

Table 1. Classification of the existing work on agent patterns.

	Domain-Independent	*Domain-Specific*
Agents	Agent architecture [12], implementation of weak agents [21], representational [12]	
Systems	Task [2], communication [8], travelling [2], coordination [9], organizational [15]	Manufacturing [22], electronic commerce [13, 5, 27], security [18]
Process	Macro-micro [25], pattern mining [28], categorization [16], application [28], vocabulary [18]	

Aridor and Lange [2] describe a set of domain-independent patterns for the design of mobile agent systems. They classify mobile agent patterns into travelling, task and interaction patterns. Kendall et al. [12] use patterns to capture common building blocks for the architecture of agents. They integrate these patterns into the Layered Agent pattern, which serves as a starting point for a pattern language for agent systems based on the strong notion of agency. Schelfthout et al. [21], on the other hand, document agent implementation patterns suitable for developing weak agents.

Deugo et al. [9] identify a set of patterns for agent coordination, which are, again, domain-independent. They classify agent patterns into architectural, communication, travelling and coordination patterns. They also describe an initial set of global forces that push and pull solutions for coordination. Kolp et al. [15] document domain-independent organizational styles for multi-agent systems using the Tropos methodology. On the other hand, Kendall [13] reports on work on a domain-specific pattern catalogue developed at BT Exact. Several of these patterns are described in the ZEUS Agent Building Kit documentation [5] using role models.

Shu and Norrie [22] and Weiss [27] have also documented domain-specific patterns, respectively for agent-based manufacturing and electronic commerce. However, unlike most other authors, they present the patterns in the form of a pattern language. This means that the relationships between the patterns are made explicit in such a way that they guide a developer through the process of designing a system.

Lind [16] and Mouratidis et al. [18] suggest that we can benefit from integrating patterns with a development process, while Tahara et al. [25], and Weiss [28] propose pattern-driven development processes. Lind [16] suggests a view-based categorization scheme for patterns based on the MASSIVE methodology. Mouratidis et al. [18] document a pattern language for secure agent systems that uses the modelling concepts of the Tropos methodology. Tahara et al. [25] propose a development method based on agent patterns and distinguish between macro and micro architecture patterns. Weiss [28] documents a process for mining for, and applying, agent patterns.

Our study of the existing literature on agent patterns leads us to conclude that these patterns pre-suppose that a decision to follow an agent approach has already been made. They focus on *how* to build agent-based systems and not on *why* to use agents. Where such reasons have been documented in the form of patterns (e.g. in [27]), they were not the main focus of the paper. It is, thus, the goal of this paper to document a generic agent pattern language as a starting point for more specific agent pattern languages. It aims to document the forces involved in agent-based design and key agent

concepts. While we realize that there are many non-technical reasons for using agents (e.g. their business benefits), our discussion is specific to technical design issues.

3 Template for Pattern Languages

As the overview of related work has shown, most agent patterns are documented in the form of pattern catalogues. Usually, the patterns are loosely related, but there is a lack of cohesion between them. Such collections of patterns provide point solutions to particular problems but do not guide the developer through the process of designing a system using those patterns. This can be achieved by a pattern language.

We argue that agent pattern authors have to put more emphasis on organizing their patterns in the form of pattern languages for them to become truly useful. In a pattern language, each pattern occupies a position in a network of related patterns, in which each pattern contributes to the completion of patterns "preceding" it in the network and is completed by patterns "succeeding" it (referred to as the *generative quality* of patterns in [3]). In the following we suggest a template for pattern languages. But first, we need to establish which qualities we are looking for in a pattern language.

Unlike a pattern catalogue that classifies patterns into categories, the goal of a good pattern language is, foremost, to create *cohesion* among the patterns. We want the patterns to be closely related to each other. References to patterns should therefore largely be to other patterns in the *same* pattern language; and the patterns should be organized from higher-level to lower-level patterns in a refinement process. We can also expect a pattern language to have a reasonable degree of *coverage* of its application domain. We want to be able to generate most of the possible designs. Finally, the goal of a pattern language is to make the links between the patterns easy to use and understandable. This we refer to as the *navigability* of a pattern language.

We can now define our template. A pattern language should contain:

- A roadmap of the pattern language
- A set of global forces
- References to other patterns in the pattern language in the context and resulting context sections of each pattern
- A discussion in each pattern of how it resolves global forces

The *roadmap* shows the structure of the pattern language. The arrows in the roadmap point from a pattern to a set of patterns that system designers may want to consult next, once this pattern has been applied. It is also often useful to identify the forces that need to be resolved in the design of the systems targetted by the pattern language. These *global forces* establish a common vocabulary among the patterns and can be used to summarize their contributions.

The *context* section of each pattern in the pattern language describes a (specific) situation in which the pattern should be considered. In particular, the context includes references to other patterns in the language in whose context the pattern can be applied ("You are using pattern X and now wish to address concern Y"). More than just

referring to "related patterns" (usually external to this set of patterns), the *resulting context* section of each pattern similarly refers to other patterns in the same language that should be consulted next, together with a rationale (that is, the trade-off addressed) for each choice ("Also consult pattern X for dealing with concern Y").

4 Pattern Language for Motivating an Agent-Based Approach

4.1 Roadmap

The structure of our pattern language is shown in the roadmap in Figure 1. It shows the patterns (rounded rectangles) in our pattern language, and their dependencies. The starting point (root) for the pattern language is AGENT SOCIETY[1]. This pattern depends on AGENT AS DELEGATE, AGENT AS MEDIATOR, and COMMON VOCABULARY. The nature of these dependencies (that is, the rationale for applying any of these patterns next) is documented in the related context section of the AGENT SOCIETY pattern. The AGENT AS DELEGATE pattern, in turn, leads us to consider USER AGENT and TASK AGENT next, just as AGENT AS MEDIATOR leads us to apply RESOURCE AGENT.

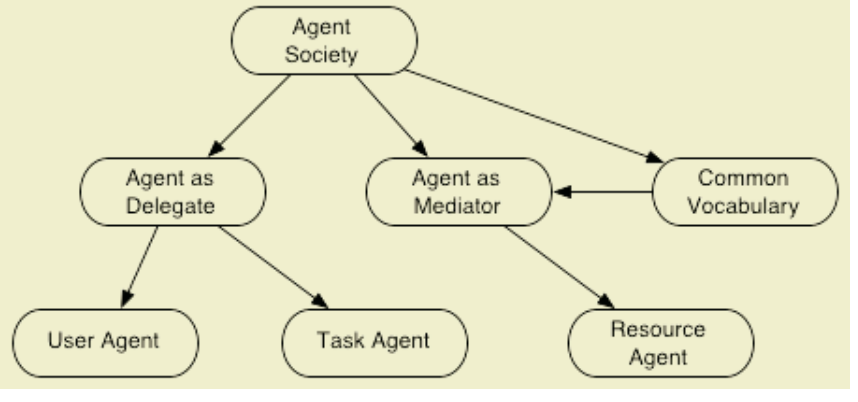

Fig. 1. Roadmap for the pattern language.

4.2 Global Forces

Next we consider the global forces involved in agent-based design. Although we lack the space for a full discussion, we can identify the following trade-offs:

- Autonomy of agents vs. their need to interact
- User's information overload vs. the degree of control the user has over agents acting on his behalf (restricting their autonomy)

[1] In the text, we set pattern names in SMALL CAPS according to the convention in [1]. In a book, we would also include the number of the page on which the pattern can be found.

- Openness of an agent society vs. its dynamicity and heterogeneity
- Need for intermediaries to facilitate agent interactions (e.g. to enforce coordination protocols) vs. privacy concerns (one form of trust)
- Heterogeneity vs. concern about quality (another form of trust)

The patterns themselves will elaborate on these trade-offs in more detail. However, it is important to note that none of these trade-offs are domain-specific.

4.3 Example

As a motivating example for using the pattern language, consider the design of a system for filtering news. This system needs to monitor a number of news sources that are distributed across the network (such as web sites, headline feeds and mailing lists) and use a variety of representation formats (e.g. HTML, RSS). The system must support both one-shot searches, as well as periodically repeated searches. The system should also filter search results based on their relevance to the individual user, thereby reducing the number of news items presented to the user to a manageable amount.

This example raises a number of design issues that agents are well suited to address. Referring to our list of global forces, we can identify information overload (of the user) as one of them. There is thus a need to delegate the task of monitoring the news sources (autonomy), as well as need to make search results highly relevant to the user by tailoring them to the user's profile. However, as a result of storing personal information in the agents, we need to protect the user's privacy.

Another set of issues stems from the fact that there are an ever-expanding number of news sources and we want to assist the user in selecting news sources relevant to their interests. This means both that the system must be open to integrate new news sources and that it must provide means for assessing the quality of news sources and their relevance to an individual user's needs. Furthermore, we need to handle the variety of representation formats used by those news sources. We don't expect to support new formats a priori, but need an architecture that simplifies their integration.

In Section 5 we will revisit this example and illustrate how the patterns described in this section can be applied to design an architecture for addressing these issues.

4.4 Patterns

The patterns of our pattern language have the following structure: pattern name, context, problem, forces, solution and resulting context.

AGENT SOCIETY

Context
Your application domain satisfies at least one of the following criteria: your domain data, control, knowledge or resources are decentralized; your application can be naturally thought of as a system of autonomous cooperating entities; or you have legacy components that must be made to interoperate with new applications.

Problem
How do you model systems of autonomous cooperating entities in software?

Forces
- The entities are autonomous in the sense that they do not require the user's approval at every step of executing their tasks but can act on their own.
- However, they rely on other entities to achieve goals that are outside their scope or reach and need to cooperate with each other.
- They also need to coordinate their behaviours with those of others to ensure that their own goals can be met, avoiding interference with each other.

Solution
Model your application as a society of agents. Agents are autonomous computational entities (autonomy), which interact with their environment (reactivity) and other agents (social ability) in order to achieve their own goals (proactiveness). Often, agents will be able to adapt to their environment and have some degree of intelligence, although these are not considered mandatory characteristics. These computational entities act on behalf of users or groups of users [17]. Thus agents can be classified as *delegates*, representing a single user and acting on her behalf, or *mediators*, acting on behalf of a group of users, facilitating between them.

The key differentiator between agents and objects is their *autonomy*. Autonomy is here used in an extended sense. It not only comprises the notion that agents operate in their own thread of control but also implies that agents are long-lived (they execute unattended for long periods), they take initiative (they do not simply act in response to their environment), they react to stimuli from the environment as guided by their goals (the *receiving* agent decides whether and how to respond to a stimulus) and they interact with other agents to leverage their abilities in support of their own as well as collective goals. *Active objects*, on the other hand, are autonomous only in the first of these senses. They are not guided by individual and/or collective goals.

A society of agents can be viewed from two dual perspectives: either a society of agents emerges as a result of the interaction of agents; or the society imposes constraints and policies on its constituent agents. Both perspectives, which we can refer to as micro and macro view of the society, respectively, mutually reinforce each other, as shown in Figure 2. Specifically, emerging agent specialization leads to the notion of roles. Roles, in turn, impose restrictions on the possible behaviours of agents [10].

This suggests two approaches to systematically designing agent societies. In the first approach, we identify top-level goals for the system and decompose them recursively until we can assign them to individual agents (as exemplified by the Gaia methodology [30]). In the second approach, we construct an agent society incrementally from a catalogue of interaction patterns, as exemplified by [13]. These interaction patterns are described in terms of roles that agents can play and their interactions, and may also specify any societal constraints or policies that need to be satisfied.

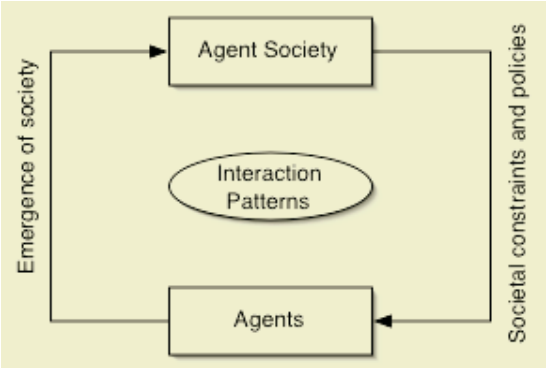

Fig. 2. Micro-macro view of an agent society.

Roles are *abstract* loci of control [13, 10, 20]. Protocols (or patterns of interaction) describe the way the roles interact. Policies define constraints imposed by the society on these roles. As an example of a policy, consider an agent-mediated auction, which specifies conventions specific to its auction type (for example, regarding the order of bids; ascending in an English auction, descending in a Dutch auction) that participating agents must comply with in order for the auction to function correctly.

Roles and their subtypes can be documented in a role diagram, using the notation introduced in [13]. *Role diagrams* are more abstract than class diagrams. Each role in the diagram defines a position and a set of responsibilities. A role has collaborators – other roles it interacts with. Arrows between roles indicate dynamic interactions between roles; the direction of an arrow represents the direction in which messages are sent between the roles. The triangle indicates a subtyping relationship between roles; subtypes inherit the responsibilities of their parent roles.

Many types of applications, such as call control [19], groupware [26] or electronic commerce applications [27], can be modelled using user, task, service and resource roles, together with their subtypes. The *user* role encapsulates the behaviour of managing a user's task agents and controlling access to the user's data. The *task* role represents users in a specific task. This is typically a long-lived, rather than one-shot, transaction. Agents in the *service* role typically provide a service to a group of users. They mediate the interaction between two or more agents through this service. The *resource* role abstracts information sources. These could be legacy data sources wrapped by "glue" code that converts standardized requests to the API of the data source.

Resulting Context

- For members of an agent society to understand each other, they need to agree on common exchange formats, as described in COMMON VOCABULARY.
- For agents that represent a single user, consult AGENT AS DELEGATE next.
- For the motivation of agents that provide services to a group of users, and their respective agents, refer to AGENT AS MEDIATOR.

AGENT AS DELEGATE

Context

You are designing your system as a society of autonomous agents using AGENT SOCIETY and you wish to delegate a single user's time-consuming, peripheral tasks.

Problem

How do you instruct agents on what to do? How much discretion (authority) should you give to an agent? How do agents interact with their environment?

Forces

- Repetitive, time-consuming tasks should be delegated to agents that can perform the tasks on behalf of their users and require only minimal intervention.
- However, when delegating a task to an agent, users must be able to trust the agent to perform the task in an informed and unbiased manner.
- The user also wants to control what actions the agent can perform on the user's behalf and which it cannot (its degree of autonomy).
- Agents need to hold information about the user but users want to control, on an interaction-by-interaction basis, which information is conveyed to another party.

Solution

Use agents to act *on behalf* of the user performing specific tasks. The structure of this pattern is shown in Figure 3. User agents manage a set of task agents, one for each task, and control access to the user's data. Task agents represent the user in different task contexts. For example, in the call control domain, a user placing, and a user receiving a call could both be represented as task agents. Each Concrete Task is a subtype of the generic Task role. The generic role contains beliefs and behaviours common to all concrete tasks. In the e-commerce domain, we would have a Trader role, plus Buyer and Seller roles that share the common belief of a desired price.

Resulting Context

- For organizing the interaction with the user to gather their requirements and feedback on the performance of a task, consult USER AGENT[2].
- Also consult USER AGENT for measures to control access to user data.
- For the design of task agents, consult TASK AGENT.

AGENT AS MEDIATOR

Context

You are designing your system as a society of autonomous agents using AGENT SOCIETY and you wish to facilitate between a group of users and their agents.

---

[2] Descriptions of USER AGENT and TASK AGENT have been omitted from this paper for space restrictions, but will be included in a future version of this language.

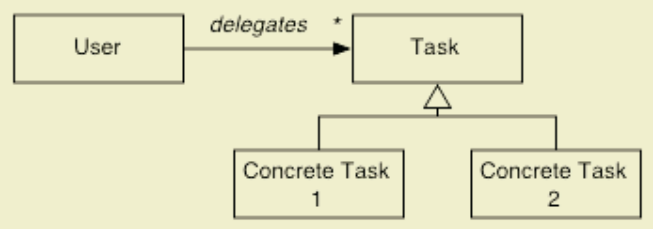

Fig. 3. Role diagram for AGENT AS DELEGATE.

Problem
How can agents find each other and coordinate their behaviours?

Forces
- In a closed agent society of known composition, agents can maintain lists of acquaintances with whom they need to interact (to obtain data or services).
- However, in an open agent society, whose composition changes dynamically, agents need help locating other agents with which they can interact.
- Agents that have no mutual history of interaction may need the help of trusted intermediaries to protect sensitive data and ensure service quality.
- Sometimes, the agents do not simply need to locate each other, but their interaction needs to follow a coordination protocol (for example, in an auction).
- A special case is that agents need to gain access to relevant resources, which creates the need for intermediaries that can find and forward queries to resources.

Solution
Use a service agent to mediate between the members of a group of agents. Examples of service agents are directories, translators, market makers and rating services. We distinguish two cases: task agents that need to locate other task agents and task agents that need to gain access to relevant resources [14, 26]. The service agent can either just pair up agents with each other (task agent-task agent or task agent-resource agent) or coordinate their interactions beyond the initial introduction.

The structure of this pattern is shown in Figure 4.

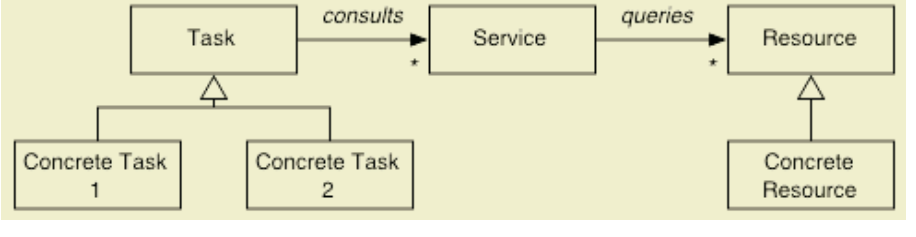

Fig. 4. Role diagram for AGENT AS MEDIATOR.

In [14], service agents are referred to as middle-agents. Different types of middle-agents can be distinguished based what services they provide. Basic mediation services comprise matching agents based on search criteria and translation services. Interaction services include the capability to coordinate the behaviours of task agents according to given protocols, conventions and policies, for example, the rules of an auction. Finally, reliability services comprise trustworthiness (of the service agent itself) and quality assurance (of the mediated services and data, as well as of the service agent).

Resulting Context

- The Agent as Mediator pattern is a starting point for many specific agent patterns, such as for search agents, recommender systems or auctions.
- For the interaction with external resources consult RESOURCE AGENT[3].

COMMON VOCABULARY

Context

When agents in an AGENT SOCIETY interact, they need to agree on common exchange formats. One scenario is that you are using agents to represent users in individual, long-living transactions as described in TASK AGENT. These task agents (for example, buyer and seller agents) need to understand each other in order to exchange messages with one other (for example, when negotiating about a price). Another scenario arises from the need to interact with resources that use a different internal representation.

Problem

How do you enable agents (for example, task agents) to exchange information?

Forces

- Agents may use different internal representations of concepts.
- To exchange information, agents need to agree on common exchange formats.
- However, common exchange formats must be widely adopted to be useful.
- If an agent needs to use multiple exchange formats to interact with different agents, it may not be able to perform all the required mappings itself.

Solution

For agents to understand each other, they need to agree on a common message format that is grounded in a common *ontology*. The ontology defines concepts that each party must use during the interaction, their attributes and valid value ranges. The purpose of this ontology is agent interaction; it does not impose any restrictions on the internal representations of the agents. In a heterogeneous, open environment, agents may even need to use multiple ontologies to interact with different agents.

The structure of this pattern is shown in Figure 5.

---

[3] A description of the RESOURCE AGENT pattern has been omitted from this paper for space restrictions, but will be included in a future version of this language.

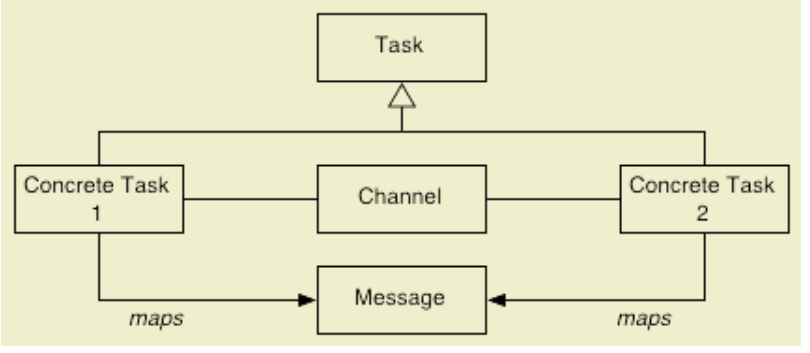

Fig. 5. Role diagram for COMMON VOCABULARY.

It is generally impractical to define general-purpose ontologies for agent interaction. These are unlikely to include the intricacies of all possible domains. Instead, the common ontology will be application-specific. Given such a shared ontology, the communicating agents need to map their internal representations to the shared ontology. Much progress has been made on XML-based ontologies, for example, in the e-commerce domain xCBL, cXML and RosettaNet are quite popular [4].

If agents need to interact with many agents using different common ontologies, it becomes impractical for the agent to be aware of all the different mappings. In this case, the need for translation agents arises that can map between ontologies on behalf of other agents. Fortunately, these are relatively straightforward to build using XSLT, a language for transforming XML documents into other XML documents.

Resulting Context

- If agents need to interact with many agents using different common ontologies, apply AGENT AS MEDIATOR to the construction of translation agents.

5 Applying the Patterns

To restate the goal of our approach, it is to use the patterns in our pattern language as guidelines for assessing whether a particular development project can benefit from an agent-based approach. However, our approach is more specific than a general discussion of the benefits of agents that can be found elsewhere. As pointed out earlier, these discussions do not provide enough information on the design trade-offs involved in migrating from a traditional to an agent-oriented system architecture.

We claim that our pattern language provides an effective way of guiding non-agent experts through the transition to an agent-based approach. The first step to applying the pattern language is to identify which of the global forces (see Section 4.2) is present in our target application domain. Then we can consult the pattern descriptions (specifically, the forces section) to see if any of the trade-offs between forces can be

resolved by a pattern. Once we have selected a pattern we should also consult the patterns referred to in its context and resulting context sections.

To give a concrete illustration of the use of this pattern language, we will apply it to the design of an architecture for the motivating example – a system for filtering news – in Section 4.4. In the description of the example, we identified several key design issues (Table 2) that our architecture will need to address but we were careful not to forgo any design decisions that already anticipate an agent-based solution.

Table 2. Global Forces in Target Application Domain.

Design Issue	*Expression in the Example*
Information overload	Users need to monitor a number of news sources distributed across the network, whose content is updated frequently.
Autonomy	Users wish to delegate the monitoring tasks to the system to be designed.
Openness	There are an ever-expanding number of news sources.
Quality	The news sources should be of high quality, and relevant to the user.
Privacy	Personal information (user's interests) needs to be stored by the system.
Heterogeneity	News sources use a variety of representation formats.

Approaching the design from a user perspective, we first consider the issue of information overload. Users are faced with a large number of news sources to which they can subscribe. For example, if we were monitoring daily news stories, there would be agencies such as Reuters and Associated Press, as well as any number of newspapers, radio and television stations that provide headline feeds. Repeatedly checking a news source for updates in a particular news category or on a particular story is a repetitive, time-consuming task that we could delegate to the system.

If we delegate the monitoring task to the system, we need to provide it with information about the users such as their preferred news categories and specific news stories they want to follow. This raises some privacy concerns. The trade-off between information overload and privacy is addressed by the AGENT AS DELEGATE pattern. It suggests representing users in different task contexts by task agents. This leads us to create a concrete task role named Query for each news category and particular story in the user's profile (together, these pieces of information form a search query).

The task agents for one user are managed by its user agent, which presents a single point of access to the system for the user. Figure 6 shows the initial architecture.

Inspecting the context and resulting context sections of AGENT AS DELEGATE suggests consulting the higher-level pattern AGENT SOCIETY, as well as the lower-level patterns USER AGENT and TASK AGENT. Regarding AGENT SOCIETY, we note that our application domain satisfies several of the criteria specified in the context of the pattern (decentralized domain data, control, resources and legacy components). Furthermore, we will be representing users as autonomous entities. From the pattern we also learn that our news sources should be represented as resource agents.

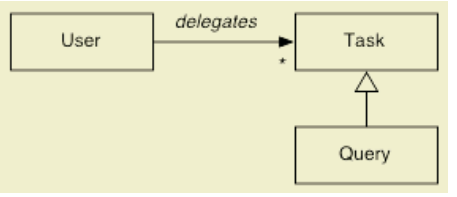

Fig. 6. Initial architecture after applying AGENT AS DELEGATE.

At this point we have two design options: task agents can interact with resource agents directly or via a service agent. The first option assumes that users are aware of all news sources relevant to them and can instruct the task agents appropriately when they are created. However, due to the openness of the system, new news sources can be added to the system at any time and assessing the relevance of those news sources becomes increasingly unmanageable. The second option, which addresses this problem, is to use service agents to mediate between task agents and resource agents.

As suggested by the resulting context of AGENT SOCIETY, we now consult AGENT AS MEDIATOR. This pattern suggests the use of an intermediary to help task agents locate news sources of relevance to the user and assess the relevance of specific news items. We can have different degrees of mediation. The first option is for the service agent to provide the locations of resource agents. Task agents then interact with these resources directly. The second option is that only the service agent interacts with the news sources. Task agents subscribe for specific news categories and particular news stories and are notified by the service agent when updates are available.

The third option is an extension of the second option. Instead of forwarding all news items received from news sources, the service agent only notifies task agents of updates that meet certain quality standards in order to make the news items more relevant to users. One way of providing quality assurance is to use collaborative filtering to recommend news items[4]. Only items that received a high relevance rating from other users will be sent to the task agents. User feedback is collected in the user agents. The rated news items are attached to the search query that produced the item.

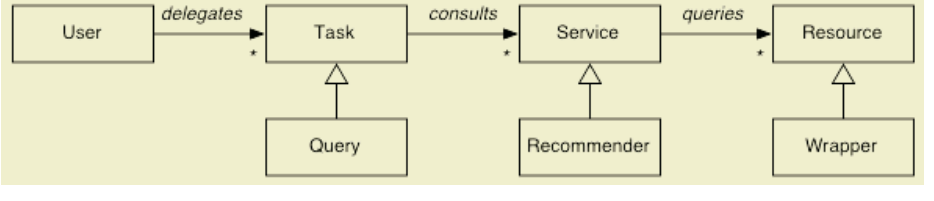

Fig. 7. Architecture after applying AGENT AS MEDIATOR.

The refined architecture illustrating the third option is shown in Figure 7. This diagram introduces two new roles. Recommender is a concrete service role correspond-

[4] Note that AGENT AS MEDIATOR by itself does not make specific suggestions such as the use of collaborative filtering but is a starting point for more specific mediation patterns.

ing to option three. Wrapper is a concrete resource role to represent agents that extract news items from news sources in a common format. There will be different types of wrappers. Each time a news source is added, a new wrapper may need to be defined, due to the heterogeneous nature of news sources. The COMMON VOCABULARY pattern should be consulted for a discussion of the design trade-offs and a general solution.

6 Conclusion

There are three main take-home messages from this paper:

- As the advantages of the agent-based approach are still not widely recognized outside our community, we need to educate non-experts in its use.
- We need guidelines for non-agent technology experts that summarize the key benefits of the agent approach; agent patterns can provide that guidance.
- This pattern language at the centre of this paper is intended to provide such guidelines and serve as a starting point for more specific pattern languages.

We also urge authors of agent patterns to organize their patterns in the form of pattern languages. To this end, a template for pattern languages has been provided.

References

1. Alexander, C., A Pattern Language, Oxford University Press, 1977
2. Aridor, Y., Lange, D., Agent Design Patterns: Elements of Agent Application Design, Second Intl. Conference on Autonomous Agents, IEEE, 1998
3. Beck, K., and Johnson, R., Patterns Generate Architectures, European Conference on Object Oriented Programming (ECOOP), 139-149, 1994
4. Carlson, D., Modeling XML Applications with UML : Practical e-Business Applications, Addison-Wesley, 2001
5. Collis, J., and Ndumu, D., The ZEUS Role Modelling Guide, BT Exact, 1999
6. Coplien, J., Software Patterns, SIGS Books, 1996
7. Deugo, D., and Weiss, M., A Case for Mobile Agent Patterns, Mobile Agents in the Context of Competition and Cooperation (MAC3) Workshop Notes, 19-22, 1999
8. Deugo, D., Oppacher, F., Ashfield, B., Weiss, M., Communication as a Means to Differentiate Objects, Components and Agents, Technology of Object-Oriented Languages and Systems Conference (TOOLS), IEEE, 376-386, 1999
9. Deugo, D., Weiss, M., and Kendall, E., Reusable Patterns for Agent Coordination, in: Omicini, A., et al. (eds.), Coordination of Internet Agents, Springer, 2001
10. Ferber, J., Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, Addison-Wesley, 13-16, 1999
11. Fernandez, E., and Pan, R., A Pattern Language for Security Models, Conference on Pattern Languages of Programming (PLoP), 2001

12. Kendall, E., Murali Krishna, P., Pathak, C., et al., Patterns of Intelligent and Mobile Agents, Conference on Autonomous Agents, IEEE, 1998
13. Kendall, E., Role Models: Patterns of Agent System Analysis and Design, Symposium on Agent Systems and Applications/Mobile Agents (ASA/MA), ACM, 1999
14. Klusch, M., and Sycara, K., Brokering and Matchmaking for Coordination of Agent Societies: A Survey, in: Omicini, A., et al. (eds.), Coordination of Internet Agents, Springer, 2001
15. Kolp, M., Giorgini, P., and Mylopoulos, J., A Goal-Based Organizational Perspective on Multi-Agent Architectures, Workshop on Agent Theories, Architectures, and Languages (ATAL), 2001
16. Lind, J., Patterns in Agent-Oriented Software Engineering, Workshop on Agent-Oriented Software Engineering (AOSE), 2002
17. Maes, P., Agents that Reduce Work and Information Overload, Communications of the ACM, 31-41, July 1994
18. Mouratidis, H., Giorgini, P., Schumacher, M., and Weiss, M., Integrating Security Patterns in the Development of Secure Agent-Based Systems, submitted, 2003
19. Pinard, D., Gray, T., Mankovski, S., and Weiss, M., Issues in Using an Agent Framework for Converged Voice and Data Applications, Conference on Practical Applications of Agents and Multi-Agents (PAAM), 1997
20. Riehle, D., and Gross, T., Role Model Based Framework Design and Integration, Conference on Object-Oriented Programs, Systems, Languages, and Applications (OOPSLA), 1998
21. Schelfthout, K., Coninx, T., et al., Agent Implementation Patterns, OOPSLA Workshop on Agent-Oriented Methodologies, 2002
22. Shu, S., and Norrie, D., Patterns for Adaptive Multi-Agent Systems in Intelligent Manufacturing, Intl. Workshop on Intelligent Manufacturing Systems (IMS), 1999
23. Silva, A., and Delgado, J., The Agent Pattern, European Conference on Pattern Languages of Programming and Computing (EuroPLoP), 1998
24. Sycara, K., Multiagent Systems, AI Magazine, 79-92, Summer 1998
25. Tahara, Y., Oshuga, A., and Hiniden, S., Agent System Development Method Based on Agent Patterns, Intl. Conference on Software Engineering (ICSE), ACM, 1999
26. Voss, A., and Kreifelts, T., SOaP: Social Agents Providing People with Useful Information, Conference on Supporting Groupwork (GROUP), ACM, 1997
27. Weiss, M., Patterns for e-Commerce Agent Architectures: Using Agents as Delegates, Conference on Pattern Languages of Programming (PLoP), 2001
28. Weiss, M., Pattern-Driven Design of Agent Systems: Approach and Case Study, Conference on Advanced Information System Engineering (CAiSE), Springer, 2003
29. Wooldridge, M., and Jennings, N., Intelligent Agents: Theory and Practice, The Knowledge Engineering Review, 10(2):115-152, 1995
30. Wooldridge, M., Jennings, N., et al., The Gaia Methodology for Agent-oriented Analysis and Design, Journal Autonomous Agents and Multi-Agent Systems, 2002

A Practical Agent-Based Approach to Requirements Engineering for Socio-technical Systems

Paolo Bresciani[1] and Paolo Donzelli[2]

[1] ITC-irst
Via Sommarive 18, I-38050 Trento-Povo, Italy
bresciani@itc.it
[2] Department of Computer Science – University of Maryland
College Park - MD, USA
donzelli@cs.umd.edu

Abstract. More powerful and pervasive information and communication technologies increasingly result into systems with a high organizational impact, which introduce organizational procedures and structures that could not exist otherwise. The software system and its application context form a larger *socio-technical* system that the requirements engineers need to analyze and understand as a whole. The overall needs of such a systems are the one to be fulfilled, while dealing with and taking into account the needs, the objective, and the expectations of a large number of stakeholders. In such a perspective, this paper introduces an agent-based requirements engineering framework, and illustrates how it has been applied throughout an eGovernment project.

1 Introduction

Information and communication technologies (ICT) are increasingly becoming more powerful and pervasive. Powerful, as they open huge improvement opportunities for organizations, by providing new ways of organizing working procedures and business processes, dealing with partners, and reaching out for customers. Pervasive, as they impact a large number of stakeholders, within an organization, by affecting organizational roles and procedures traditionally left untouched by the ICT trend.

Such combination of power and pervasiveness results into systems with a high organizational impact. These systems do not simply aim at providing more efficient ways of performing old tasks (e.g., by replacing one or more existing systems, or automating well-established procedures and routines), but, specially, at introducing new ways of working that could not exist otherwise. In other terms, such a software system and its application context can be considered to form a larger human/technological system, that have to be treated and analyzed as a whole[12]. In this sense we refer to *socio-technical* systems. Complexity of socio-technical systems goes beyond working procedures and the software

P. Giorgini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 158–173, 2004.

systems themselves: it encompasses the complexity generated by the impact of the system upon the organizational structure, from the business process, to the behavior of the single employee.

Requirements Engineering (RE) [14] deals with the definition, formalization, and analysis of the requirements that a potential system must have to accomplish organization specific needs.

In Requirements Engineering, Goal and Agent orientation [15,1,13,4,8,9,10] has been recognized as a promising approach to deal with socio-technical systems, as the needs emerging within the social and organizational context can be directly linked to the features of the technological system. By adopting the notions of *Agent*, *Goal*, and *Intentional Dependency*, in fact, it is possible to refine, in a smooth and controlled manner, the high-level organizational needs into detailed descriptions of the system to be implemented. The concepts of Agent, Goal, and Intentional Dependency, in fact, applied to describe the social setting in which the system has to operate, lead towards a smooth and natural system development process, spanning from high-level organizational needs to system deployment [4]. Goals are valuable in identifying, organizing and justifying system requirements [16,4,10,8,2,15], whereas the notion of agent [16,4,10,2,9] provides a quite flexible mechanism to model the stakeholders.

However, the concrete application of such approaches has been until now limited only to few case studies. Several causes of this still immature adoption of agent- and goal-based paradigms for RE may be identified. Below we consider only two of them.

First, although the notion of goal is central in some RE consolidated approaches like ***i**** [16], GBRAM [1,2], and KAOS [8], an integrated and comprehensive requirements analysis methodology, clearly linked – or linkable – to the subsequent phases of software development, still is an open issue. At best of our knowledge, only the Tropos methodology [5,4] fully addresses this issue. Yet, not full consideration has been given by Tropos itself to the design of a precise process for the RE phases (early requirements and late requirements), due to the wide set of aspects that have to be captured.

Second, concerning the RE component of Tropos (or ***i****, to which Tropos RE is largely inspired), it is worth noticing that its considerably rich modeling framework, although promises to be capable of capturing several aspects relevant for the following phases, it also shows a certain level of complexity, so resulting understandable to only a strict group of practitioners. When the use of an ***i**** like modeling language has to be extended to non-technical stakeholders, it may be appropriate to give up with the full language expressiveness and modeling flexibility, in favor of a more straightforward and simple way to communicate with the stakeholders.

In such a perspective, the paper introduces an agent- and goal-based RE Framework (called *REF*) previously applied to an extensive project for the definition of the requirements of a simulation environment [11]. Simple, yet reasonably expressive, REF allows non technical stakeholders to elicitate requirements, in collaboration with a requirements engineer which, at the same time, is pro-

vided with an effective methodology and process for requirements acquisition, analysis and refinement, and for communicating, in an easily intelligible way, the results of his/her analysis to the stakeholders.

The paper is organized as follows. After a brief description of the REF background (Section 2), its main characteristics are highlighted in Section 3. A case study, is adopted (Section 4) to present several aspects of the methodology, both from the point of view of the practical usability and acceptability by the stakeholders. Conclusions are given in Section 5.

2 Background

REF combines advanced requirements engineering techniques with software quality modeling approaches [3], to capture the stakeholders perception of quality since the beginning of a new project, and to produce agreeable-upon and implementable functionalities and constraints. In particular, we highlight here the aspect derived from RE techniques.

REF is strongly based upon ***i****, the modeling framework suggested by Eric Yu [16,18,17], and thus open to be integrated in – or extended by – the Tropos methodology. However, it introduces some simplifications and tends to adopt a more pragmatic approach in order to obtain a greater and more active involvement of the stakeholders during the requirements discovery, elicitation and formalization process. The aim is to let the stakeholders to easily grasp the notation since the beginning of a project. The adopted simplifications, with respects to ***i****, include:

- the use of only one type of actor/agent (i.e., no distinction is made between agent, role and position);
- the use of only one kind of link, both as dependency link as well as decomposition and means-end link. In the latter two contexts the arrowhead direction is reversed with respect to that used in ***i****.

These choices allows for a more intuitive reading of the REF diagrams, corresponding to the direction adopted by the process in developing the diagrams (top-down). As well, a more natural flow of dependencies from the dependencies among agents – in the organization model – to the decomposition dependencies – in the (hard and soft) goal models – and vice versa, is obtained.

REF adopts a strict top-down approach, during which the analysts, in close cooperation with the stakeholders, drill through the organization, until reaching the desired level of detail, by using three different (but similar) modeling tools: organization models, soft-goal models, and hard-goal models. Unlike ***i****, REF emphasizes the operational role that soft-goals can play in deriving the requirements of a new system. Soft-goals, in fact, beyond playing an important role in supporting reasoning between alternatives (as in ***i****), since the early stages of a project, can also provide a systematic and organized way of handling non-functional requirements in a very pragmatic way. In fact, REF recognizes the need of explicitly resolving soft-goals, by turning them into more manageable constraints. Thus, soft-goal modeling becomes one of the main modeling

efforts and a powerful tool, in the analysts hands, to detect and resolve clashing requirements.

3 REF

REF is designed to provide the analysts and the stakeholders with a powerful tool to capture high-level organizational needs and to transform them into system requirements, while redesigning the organizational structure to better exploit the new system.

The framework tackles the modeling effort by breaking the activity down into more intellectually manageable components, and by adopting a combination of different approaches, on the basis of a common conceptual notation.

Agents are used to model the organization [9,11,17]. The organizational context is modeled as a network of interacting agents (which represent any kind of active entity, e.g., teams, humans and machines, one of which is the target system), collaborating or conflicting in order to achieve both individual and organizational goals. *Goals* [9,11,8] are used to model agents relationships, and, eventually, to link organizational needs to system requirements. According to the nature of a goal, a distinction is made between *hard-goals* and *soft-goals*. A goal is classified as *hard* when its achievement criterion is sharply defined. For example the goal *"document be available"* is a hard-goal, being easy to check whether or not it has been achieved (i.e., is the document available, or not?). For a *soft-goal*, instead, it is up to the goal originator, or to an agreement between the involved agents, to decide when the goal is considered to have been achieved. For example, the goal *"document easily and promptly available"* is a soft-goal, given that when we introduce concepts such as *"easy"* and *"prompt"*, different persons usually have different opinions.

REF tackles the modeling effort by supporting three inter-related activities, as listed below (see also Figure 1). Such activities do not exist in isolation, rather they are different views of the same modeling effort, linked by a continuous flow of information, schematized as Development, and Elicitation & Validation flows.

Organization Modeling, during which the organizational context is analyzed, and the agents and their goals identified. Any agent may generate its own goals, may operate to achieve goals on the behalf of some other agents, may decide to collaborate with or delegate to other agents for a specific goal, and might clash on some other ones. The resulting goals will then be refined, through interactions with the involved agents, by hard- and soft-goal modeling.

Hard-Goal Modeling seeks to determine how an agent can achieve a received hard-goal, by decomposing it into more elementary subordinate hard-goals, *tasks*[1], and *resources*[2]. Supported by the REF graphical notation, the analyst and the stakeholder will work together to understand and formalize how

[1] A *task* is a well-specified prescriptive activity.

[2] A *resource* is any concrete or information item necessary to perform tasks or achieve goals.

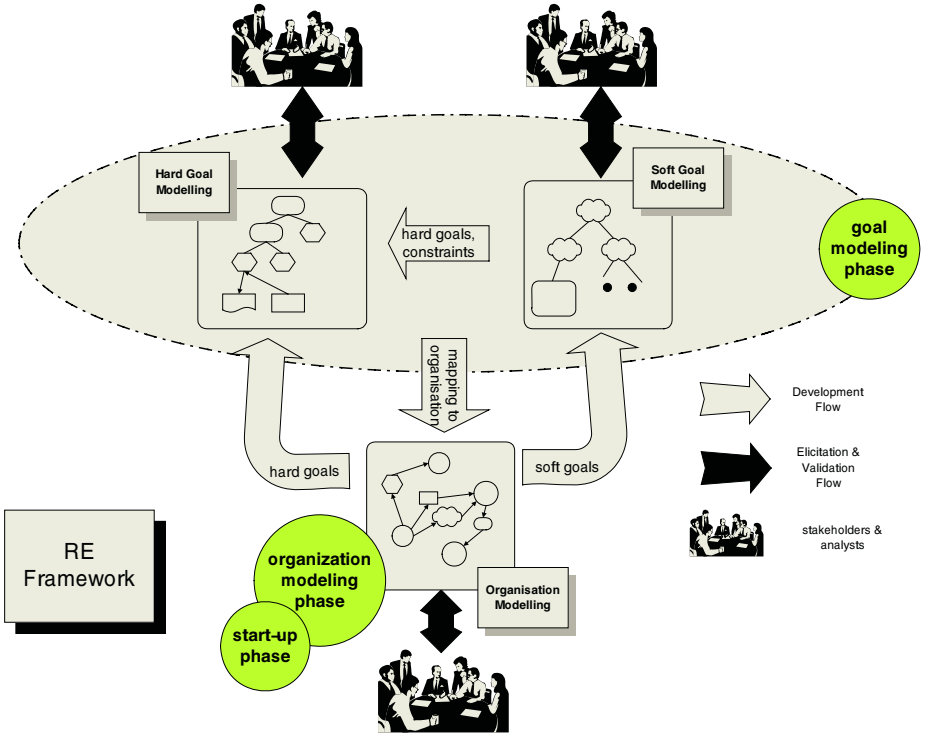

Fig. 1. The REF process.

the agent[3] thinks to achieve the goal, in terms of subordinate hard-goals and tasks that s/he will have to achieve and perform directly or indirectly, by passing them to other agents.

Soft-Goal Modeling aims at producing the operational definitions of the soft-goals, sufficient to capture and make explicit the semantics that are usually assigned implicitly by the involved agents [3,6,7]. Unlike for a hard-goal, for a soft-goal the achievement criterium is not, by definition, sharply defined, but implicit in the originator intentions. The analyst's objective during soft-goal modeling is to make explicit such intentions, in collaboration with the goal originator. However, depending on the issue at hand, and the corresponding role played by the two agents (i.e., the originator and the recipient) within the organization, also the recipient may be involved in the process, to reach a sharply defined achievement criterium upon which both of them can agree. The main aim during Soft-Goal Modeling is to iteratively refine each soft-goal in terms of subordinate elements, until only hard-goals, tasks, resources, and *constraints* are obtained (that is, until all the soft aspects have been dealt with) or each not refined soft-goal is passed on to another agent, in the context of which will then be refined. *Constraints* may be as-

[3] Let us recall that an agent may represent the stakeholder him/herself.

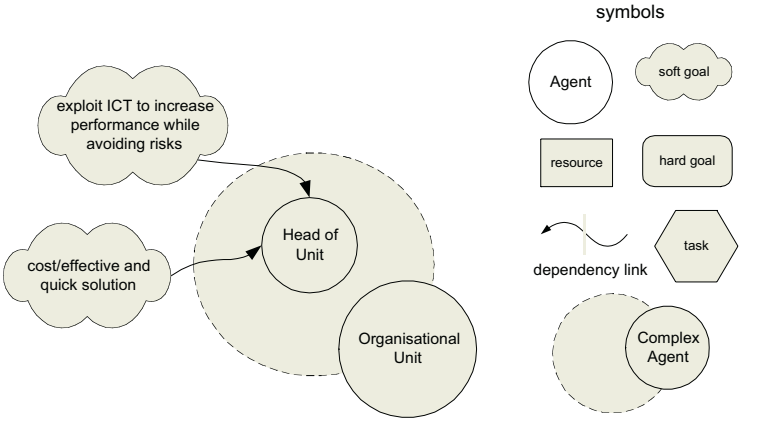

Fig. 2. Introducing the ERMS: the initial organization model.

sociated with hard-goals, tasks, and resources to specify the corresponding quality attributes. Thus, the resulting set of constraints represents the final and operationalized views of the involved quality attributes, i.e., the quality models that formalize the attributes for the specific context [3,6].

In the three modeling activities, REF uses a diagrammatic notation which immediately conveys the dependencies among different agents and allows for a detailed analysis of the goals, upon which the agents depend, in favor of a higher usability and acceptability by the stakeholders.

In the next Section the notation and the methodology is introduced by means of a case study.

4 Applying REF: A Case Study

We refer to a real project aimed at introducing an Electronic Record Management System (ERMS) within a government unit. The impact of such a system on the common practices of the communities and sub-communities of knowledge workers is quite relevant.

A ERMS is a complex ICT system which allows for efficient storage and retrieval of document-based unstructured information, by combining classical filing strategies (e.g., classification of documents on a multi-level directory, cross-reference between documents, etc.) with modern information retrieval techniques. Moreover, it provides mechanisms for facilitating routing and notification of information/documents among the users, and supporting interoperability with similar (typically remote) systems, through email and XML.

The impact of such a system on the common practices of the communities and the sub-communities of knowledge workers who will adopt it is quite relevant.

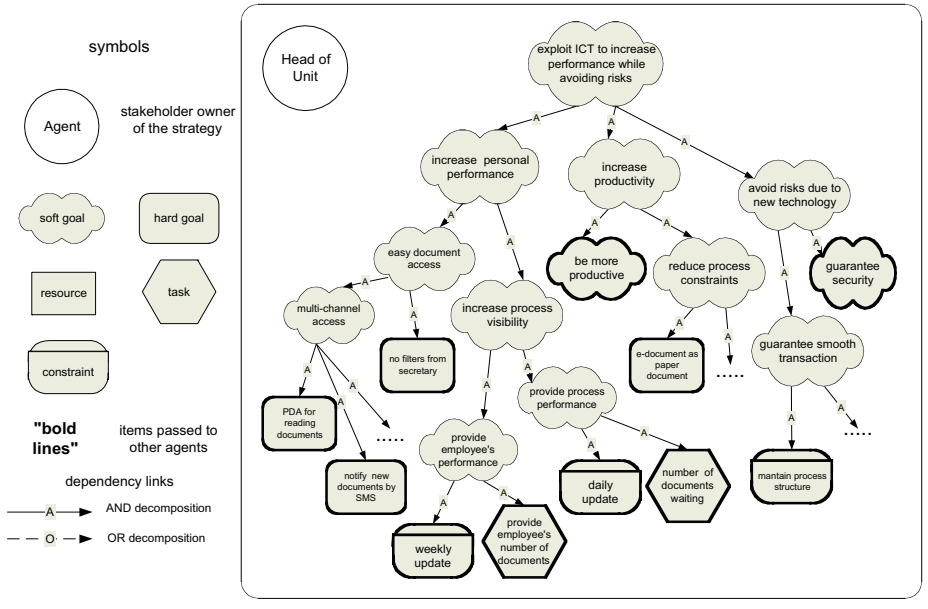

Fig. 3. The "exploit ICT to increase performance while avoiding risks" Soft-Goal Model.

4.1 The Initial Organization Model

Several factors (international benchmarking studies, citizens demand, shrunk budgets, etc.) called for the decision of leveraging new technologies to transform the organization. The initial organization model is shown in Figure 2. Circles represent agents, and dashed lines are used to bound the internal structure of complex agents; that is, agents containing other agents. In Figure 2, the complex agent Organizational Unit corresponds to the organizational fragment into which it is planned to introduce the new ERMS, whereas the Head of Unit is the agent, acting within the Organizational Unit, responsible for achieving the required organizational improvement (modeled by the soft-goals exploit ICT to increase performance while avoiding risks, and cost/effective and quick solution).

Goals, tasks, resources and agents (see also next Figures) are connected by dependency links, represented by lines with arrowheads. An agent is linked to a goal when it needs or wants that goal to be achieved; a goal is linked to an agent when it depends on that agent to be achieved. Similarly, an agent is linked to a task when it wants the task to be performed; a task is linked to an agent when the agent is committed at performing the task. Again, an agent is linked to a resource when it needs that resource; a resource is linked to an agent when the agent has to provide it. By combining dependency links, we can establish dependencies among agents. So, for example, a path $A_1 \rightarrow G \rightarrow A_2$ (where A_1 and A_2 are agents and G is a goal) means that the agent A_1 depends on the agent A_2 to achieve the goal G.

4.2 Goal Modeling

Once built the initial organization model, REF analysis proceeds by focusing on the emerging goals, in this case we focus on the soft-goal exploit ICT to increase performance while avoiding risks.

As mentioned, the soft-goals modeling process allows the analysts and the stakeholders to operationalize all the soft aspects implicitly included in the meaning of the soft-goal.

Thus, for example, Figure 3 describes how the soft-goal exploit ICT to increase performance while avoiding risks is iteratively top-down decomposed to finally produce a set of tasks, hard-goals, and constraints that precisely defines the meaning of the soft-goal, i.e., the way to achieve it. Figure 3, in other terms, represents the strategy that the Head of Unit (as result of a personal choice or of a negotiation with the upper organizational level) will apply to achieve the assigned goal. Again, the arrowhead lines indicate dependency links. A soft-goal depends on a sub-ordinate soft-goal, hard-goal, task, resource or constraint, when it requires that goal, task, resource or constraint to be achieved, performed, provided, or implemented in order to be achieved itself. These dependency links may be seen as a kind of top-down decomposition of the soft-goal. Soft-goals decompositions may be conjunctive (all the sub-components must be satisfied, to satisfy the original soft-goal), indicated by the label "A" on the dependency link, or disjunctive (it is sufficient that only one of the components is satisfied), indicated by the label "O" on the dependency link (see Figure 7).

According to Figure 3, the Head of Unit has to increase personal performance, to increase productivity of the whole unit, and also to avoid risks due to new technology. Let's consider in details only the first sub-soft-goal, i.e., increase personal performance. It spawns two subordinate soft-goals: easy document access, for which the Head of Unit will require a multi-channel access system in order to be able to check and transfer the documents to the employees also when away from the office; and increase process visibility, to take better informed decisions. In particular, the soft-goal increase process visibility will eventually lead to the identification of some tasks (functionalities) the system will have to implement in order to collect and make available some data about the process (e.g., number of documents waiting) and about the employees (provide employee's number of documents that have been assigned), and of some associated constraints, represented by a rounded-rectangle with a horizontal line, characterizing such data. In Figure 3, for example, they specify the frequency of update: daily for the process data (daily update) and weekly for the employee's ones (weekly update).

4.3 Dealing with Multiple Solutions

In the previous sub-section, a top-down expansion and analysis – as required by REF – of the soft-goal model (here represented as a tree) has been illustrated. This kind of approach may lead at introducing different sub-goals (or constraints, or tasks, or resources) for any different goal that emerges during the goal analysis activity, even if different goals could be satisfied by the same

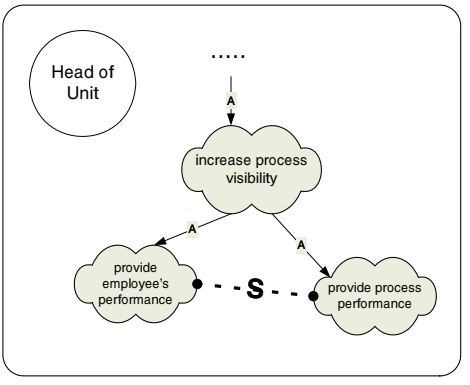

Fig. 4. A sharing between goals of the same agent.

sub-goal (or constraint, or task, or resource). For example, in Figure 3, two distinct constraints have been introduced for satisfying the two soft-goals provide process performance and provide employee's performance, namely the constraints daily update and weekly update.

An evident consequence of such an analysis is that two distinct, specific functional requirements are posed on the system-to-be.

In some cases, it could be preferable to introduce a common child for two (or more) different intermediary nodes, before proceeding with the analysis of the sub-trees. Thus leading to a common – *shared* – analysis of the remaining part of the model.

Moreover, it could be strategic to be able to recognize such an eventuality as early as possible during the modeling activity, so that possible alternative modeling strategies can be properly considered and compared. In fact, different diagram evolution strategies, and thus development sequences, may lead to quite different results or, even when producing the same result, this may be obtained with different degrees of efficiency. For example, a top-down bread-first diagram expansion could be probably preferred to a top-down depth-first strategy. In this way, it may appear appropriate to develop a shared sub-tree only once, with two advantages: 1) at the requirements definition level, the analysis has not to be carried out twice; 2) at the software development level, the complexity of the process and the system to be implemented will be reduced, being two potentially different requirements, and all the derived artifacts – from the architectural design down to the implemented code – collapsed into a single one.

For this reasons, REF allows the analysts to annotate, during their refinement and navigation through the goals and sub-goals models, situations of possible "sharing" for sub-trees (including, in particular, single nodes) where they believe that some commonalities could be hidden. This is obtained through a specific notation called the *S-connection* ("S" for Sharing), a link that does not have arrows, being the relationship symmetric. It can be used as a high-level reasoning support tool, to enable the analysts to record their intuitions while building the

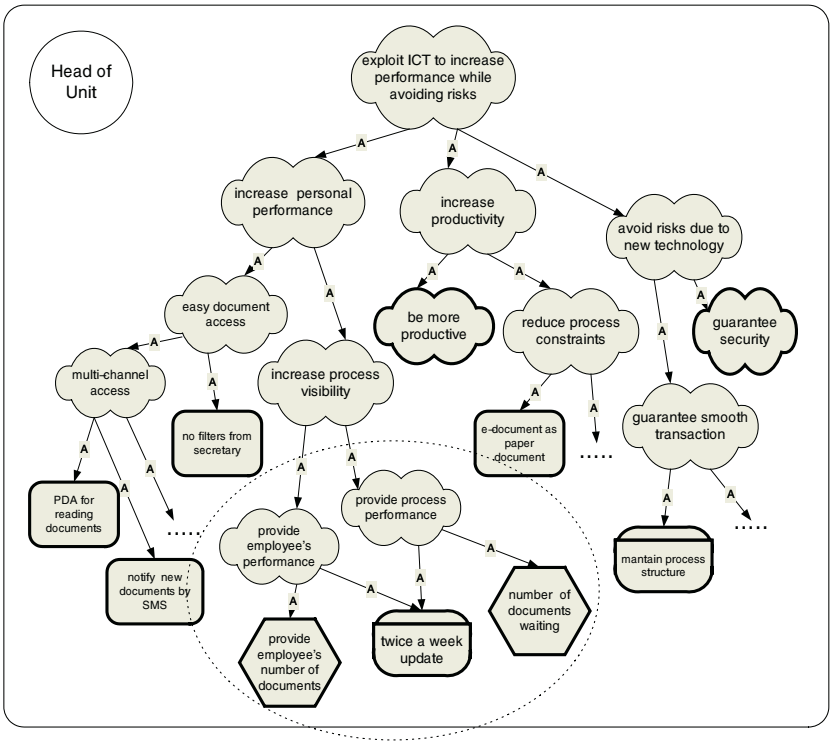

Fig. 5. The "exploit ICT to increase performance while avoiding risks" Soft-Goal Model revised.

goals models, by making notes to drive their strategies, e.g., to highlight where a top-down breath-first diagram expansion may be preferable to a top-down depth-first one.

Figure 4 represents an example of use of the S-connection.

Adopting this evolution path of the soft-goal model, instead of the result illustrated in Figure 3, the slightly different model of Figure 5 is produced. Here, the same constraint (twice a week update) is shared among the two soft-goals provide employee's performance and provide process performance.

It is worth noticing that, without the support of the S-connection, any sequence might have been followed in analyzing the two soft-goals, with the possible consequence of introducing the two different constraints (weekly update and daily update) in two very different moments, making it very difficult to spot that a common (although slightly different) constraint could have been adopted. Adopting the S-connection, instead, allows the analysts to keep clear and well planned the modeling task of evaluating, in collaboration – in this case – with the Head of Unit, the possibility of a compromise.

For sake of simplicity, the differences here presented, between Figure 3 and Figure 5, are minimal, and regard only leaf nodes, as highlighted by the dotted

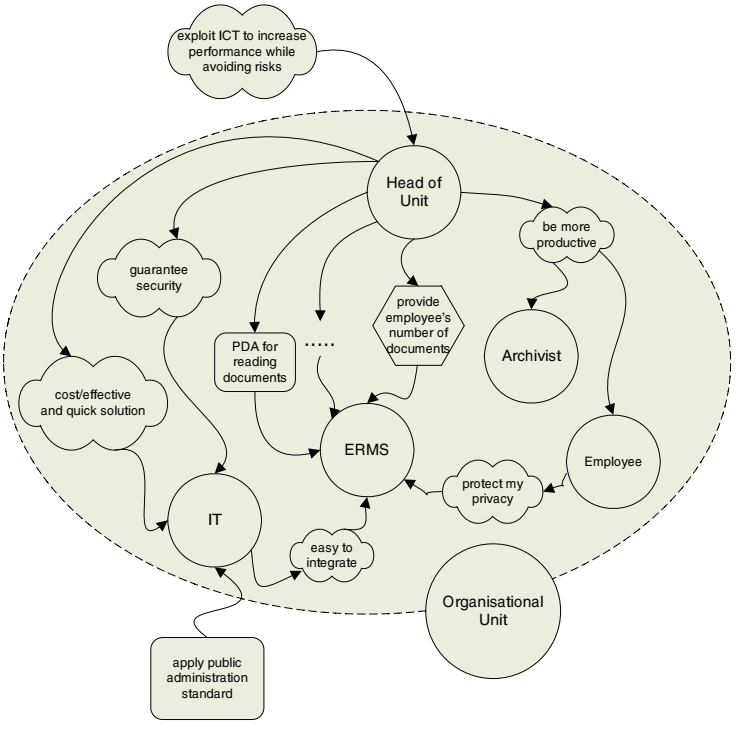

Fig. 6. Evolving the Organization Model.

circle in Figure 5. Re-designing the new model starting from the old one would not require a big effort. But in more complex cases, in which the two collapsing nodes are non-leaf nodes but have deep trees expanding from them – possibly involving different goal models even of different agents – relevant parts of the two sub-trees, rooted in the two nodes, could require considerable revisions, to be merged. The use of the S-connection is aimed at preventing late – and costly – revisions, by guiding towards a-priori, better informed, strategical choices during the modeling activity.

4.4 Evolving the Organization Model

In Figures 3 and 5, the items in bold outline are those that the Head of Unit will pass out, having decided to depend on other agents for their achievement. For such a reason, they are not further analyzed, instead they will be refined as further agreement between the Head of Unit and the agent that will be appointed of their achievements. The results of this analysis allow us to enrich the initial organization model in Figure 2, leading to the model in Figure 6, where some details have been omitted for the sake of clarity. In Figure 6, some new agents have been introduced: the Archivist and the Employee, which have to be more

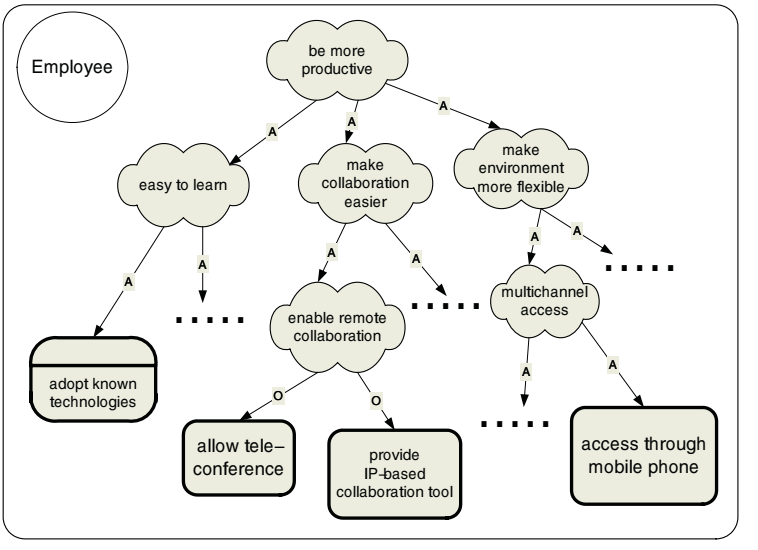

Fig. 7. The "be more productive" Soft-Goal Model.

productive, the Information Technology (IT), which has to guarantee security, and the ERMS, upon which the identified goals, tasks and constraints will be placed.

From Figure 6, we can also see that the Head of Unit has decided to delegate the soft-goal cost/effective and quick solution to the IT agent, which, on its turn, will have to achieve other goals coming from the external environment, such as, for example, apply public administration standards. At this point, the analysis can be carried on by focusing, for example, on how the Employee will try to achieve the soft-goal be more productive.

4.5 Goal Modeling Again

Here, following the REF modeling cycle, we are back to goal modeling, in particular to soft-goal modeling.

To be more productive, the Employee will define his/her own strategy, eventually reaching an agreement with the Head of Unit. Such a strategy is shown by the soft-goal model in Figure 7, where we can see how, in order to be more productive the Employee will ask that the system has to be easy to learn and has to make collaboration easier with the other employees, which are dealing with the same document. For sake of brevity, only a partial refinement of these soft-goals is shown. The soft-goal easy to learn will spawn, among other items here omitted, the constraint adopt known technologies (i.e., technologies which the employee is already used to), whereas the soft-goal make collaboration easier will lead, through further refinement, to a series of hard-goals implying specific

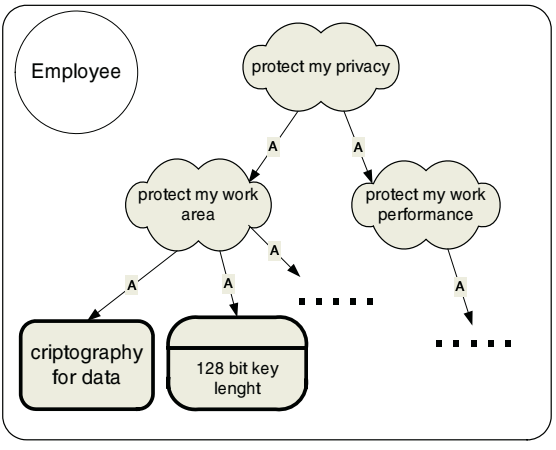

Fig. 8. The "protect my privacy" Soft-Goal Model.

capabilities (e.g., either a teleconference or an IP-based collaboration tool, see the hard-goals allow tele-conference and provide IP-based collaboration tool).

Another soft-goal model for the Employee may derive from his/her concern of possible side effects on his/her privacy, related to the introduction of the ERMS. This is represented by the soft-goal protect my privacy, posed upon the ERMS in Figure 6. By modeling the soft-goal protect my privacy (Figure 8), we can discover that the Employee wants to be sure that its own private area of work will not be accessible by others, asking for example to be enabled to adopt a cryptography tool (whit a 128 bit length security key) to protect particular data, but, above all, given that all the activities will be performed through the ERMS, that s/he does not want someone to be able to monitor his/her actions. This clearly conflicts with the possibility of monitoring the employees performance that was required by the Head of Unit (Figure 3 and Figure 5), requiring some trade-off to be identified.

4.6 Concluding Revision of the Organization Model

As a concluding remark, let us consider a last – even though, still partial – organization model, as depicted in Figure 9. Here, a slightly different model than the one presented in Figure 6 is shown: the Head of Unit's objective provide employee's performance is kept at a soft-goal level, in order to deal with possible conflicts arising from the contrasting Employee's soft-goal protect my privacy. Very similar to the S-connection – but opposite in its meaning – here a *H-connection* ("H" for hurting) is used.

This is a useful tool to highlight possible conflicts and try to reconcile different stakeholders points of view, allowing to evolve the analyses only along the most promising alternatives.

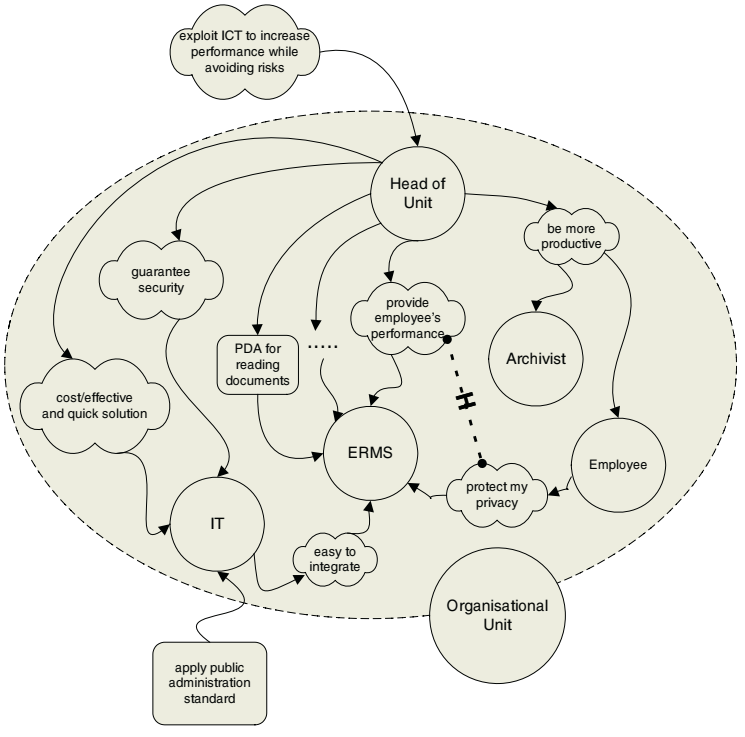

Fig. 9. A conflict between goals of different agents.

Thus, here, the possible evolution of the soft-goal model introduced in Figure 8 (in particular of the soft-goals protect my privacy and protect my work performance) leading to the constraint that the ERMS should not be allowed to monitor and record data regarding the Employee's actions, is foreseen by the analyst. This clashes with the desires expressed by the Head of Unit of knowing the number of documents each Employee is dealing with (Figure 3 and Figure 5), so that a different solution as to be found, as an agreement between the Head of Unit and the Employee, as, for example, the decision of providing the Head of Unit with only the average number of the documents dealt with by all the Employee's, so protecting the privacy of the single one (see also [10]).

5 Conclusions

The paper introduced an agent-based Requirements Engineering Framework (REF), explicitly designed to support the analysts in dealing with socio-technical systems. In particular, REF enables the analyst to capture, represent, and transform high-level organizational needs into system requirements, while redesigning the organizational structure itself. The underlying concepts and the adopted notations make of REF a very effective and easy to deal with (usable) tool, able

to tackle complex real case situations, while remaining simple enough to allow a practical and effective stakeholders involvement.

REF is strongly based upon ***i****, the modeling framework suggested by Eric Yu [16,18,17]. However, it introduces some simplifications and tends to adopt a more pragmatic approach in order to obtain a greater and more active involvement of the stakeholders during the requirements discovery, elicitation and formalization process.

The case study results show the feasibility of the proposed approach and the benefits it can provide.

References

1. A. I. Antón. Goal-based requirements analysis. In *Proceedings of the IEEE International Conference on Requirements Engineering (ICRE '96)*, Colorado Springs, USA, Apr. 1996.
2. A. I. Antón and C. Potts. Requirements for evolving systems. In *Proceedings of the International Conference on Software Engineering (ICSE '98)*, Kyoto, Japan, Apr. 1998.
3. V. R. Basili, G. Caldiera, and H. D. Rombach. *Encyclopedia of Software Engineering*, chapter The Goal Question Metric Approach. Wiley&Sons Inc, 1994.
4. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 2003. in Press.
5. P. Bresciani, A. Perini, , F. Giunchiglia, P. Giorgini, and J. Mylopoulos. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In *Proceedings of the Fifth International Conference on Autonomous Agents*, Montreal, Canada, May 2001.
6. G. Cantone and P. Donzelli. Production and maintenance of goal-oriented software measurement models. *International Journal of Knowledge Engineering and Software Engineering*, 10(5):605–626, 2000.
7. L. K. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Publishing, 2000.
8. A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.
9. M. D'Inverno and M. Luck. Development and application of an agent based framework. In *Proceedings of the First IEEE International Conference on Formal Engineering Methods*, Hiroshima, Japan, 1997.
10. P. Donzelli and P. Bresciani. Goal-oriented requirements engineering: a case study in e-government. In J. Eder and M. Missikoff, editors, *Advanced Information Systems Engineering (CAiSE'03)*, number 2681 in LNCS, pages 605–620, Klagenfurt/Velden, Austria, June 2003. Springer-Verlag.
11. P. Donzelli and M. Moulding. Developments in application domain modelling for the verification and validation of synthetic environments: A formal requirements engineering framework. In *Proceedings of the Spring 99 Simulation Interoperability Workshop*, LNCS, Orlando, FL, 2000. Springer-Verlag.
12. S. Fickas and B. Helm. Knowledge representation and reasoning in the design of composite systems. *Transactions on Software Engineering*, 18(6):470–482, June 1992.

13. J. Mylopoulos and J. Castro. Tropos: A framework for requirements-driven software development. In J. Brinkkemper and A. Solvberg, editors, *Information System Engineering: State of the Art and Research Themes*, Lecture Notes in Computer Science. Springer-Verlag, 2000.
14. A. van Lamsweerde. Requirements Engineering in the Year 00: A Research Perspective. In *Proceedings of the 22nd International Conference on Software Engineering, Invited Paper*, ACM Press, June 2000.
15. A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of RE'01 - International Joint Conference on Requirements Engineering*, pages 249–263, Toronto, aug 2001. IEEE.
16. E. Yu. *Modeling Strategic Relationships for Process Reengineering.* PhD thesis, University of Toronto, Department of Computer Science, University of Toronto, 1995.
17. E. Yu. Why agent-oriented requirements engineering. In *Proceedings of 3rd Workshop on Requirements Engineering For Software Quality*, Barcelona, Catalonia, June 1997.
18. E. Yu and J. Mylopoulos. Using goals, rules, and methods to support reasoning in business process reengineering. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 1(5):1–13, Jan. 1996.

AOR Modelling and Simulation: Towards a General Architecture for Agent-Based Discrete Event Simulation*

Gerd Wagner

Faculty of Technology Management, I&T, Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
G.Wagner@tm.tue.nl
http://www.tm.tue.nl/it/staff/gwagner/

Abstract. Agent-oriented modelling of software systems and agent-based simulation are commonly viewed as two separate fields with different concepts and techniques. We show that the *Agent-Object-Relationship (AOR)* modelling language, proposed in [Wag03] for information systems analysis and design, can also be used for specifying simulation models that can be executed by an agent-based simulation system. The suitability of AOR modelling for simulation is also supported by the fact that the AOR meta-model and the meta-model of discrete event simulation can be combined into a model of agent-based discrete event simulation in a natural way.

1 Introduction

Agent-based simulation (ABS) is a new paradigm that has been applied to simulation problems in biology, engineering, economics and sociology. In ABS, a scenario of systems that interact with each other and with their environment is being modelled, and simulated, as a *multiagent system*. The participating agents – animals, humans, social institutions, software systems or machines – can perform actions, perceive their environment and react to changes in it. They also have a mental state comprising components such as knowledge/beliefs, goals, memories and commitments.

Compared to traditional simulation methods – like mathematical equations, discrete event-simulation, cellular automata and game theory – ABS is less abstract and closer-to-reality, since it explicitly attempts to model the specific behaviour of individual actors, in contrast to macro simulation techniques that are typically based on mathematical models averaging the behaviour effects of individuals or of entire populations (see [Dav02]).

There are many different formalisms and implemented systems that are all subsumed under the title 'agent-based simulation'. For example, one of the most prominent ABS systems is SWARM [MBLA96], an object-oriented programming library that provides special support for event management, but does not support any cognitive agent concept. Like SWARM, many other ABS systems do not have a theoretical foundation in the form of a *metamodel*. They therefore do not allow the specification

* This paper improves and extends [WT03].

P. Giorgini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 174–188, 2004.

of a simulation model in a high-level declarative language but require specifying simulation models at the level of imperative programming languages.

We take a different approach and aim at establishing an ABS framework based on a high-level declarative specification language with a UML-based visual syntax and an underlying simulation metamodel as well as an abstract simulator architecture and execution model. The starting point for this research effort is an extension and refinement of the classical *discrete event simulation* paradigm by enriching it with the basic concepts of the *Agent-Object-Relationship (AOR)* metamodel [Wag03].

2 Modelling Agents and Multiagent Systems

2.1 AOR Modelling

The AOR modelling language (AORML) is based on an ontological distinction between active and passive entities, that is, between agents and (non-agentive) objects of the real world. The agent metaphor subsumes *artificial* (software and robotic), *natural* (human and animal) as well as *social/institutional* agents (groups, organizations etc.).

In AORML, an entity is an agent, an event, an action, a claim, a commitment or an ordinary object. Only agents can communicate, perceive, act, make commitments and satisfy claims. Objects are passive entities with no such capabilities. Besides human and artificial agents, AORML also includes the concept of *institutional agents,* which are composed of a number of other agents that act on their behalf. Organizations and organizational units are important examples of institutional agents.

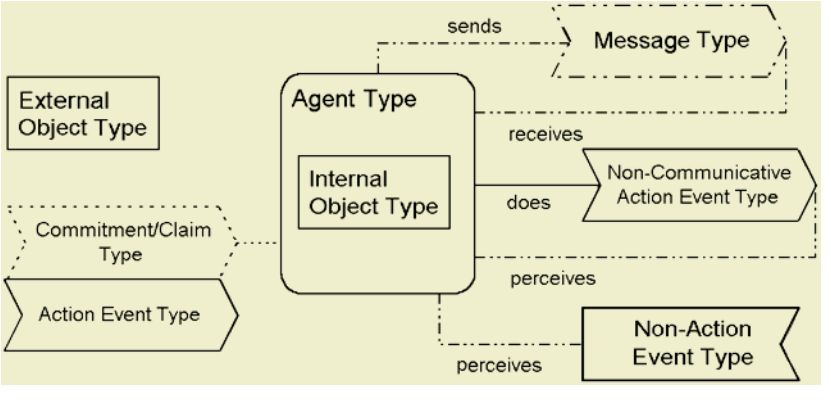

Fig. 1. The core state structure modelling elements of external AOR diagrams.

There are two basic types of AOR models: *external* and *internal* models. An external AOR model adopts the perspective of an external observer who is looking at the (prototypical) agents and their interactions in the problem domain under consideration. In an internal AOR model, we adopt the internal (first-person) view of a particular agent to be modelled. While a (business) domain model corresponds to an external model, a design model (for a specific information system) corresponds to an internal model, which can be derived from the external one.

Fig. 1 shows the most important elements of external AOR state structure modelling. There is a distinction between *action events* and *non-action events*, and between a *communicative* action event (or *message*) and a *non-communicative* action event. Fig. 1 also shows that a *commitment/claim* is coupled with the action event that fulfils that commitment (or satisfies that claim).

The most important behaviour modelling element of AORML are ***reaction rules***, which are used to express *interaction patterns*. A reaction rule is visualized as a circle with incoming and outgoing arrows drawn within the agent rectangle whose reaction pattern it represents. Each reaction rule has exactly one incoming arrow with a solid arrowhead: it specifies the triggering event type. In addition, there may be ordinary incoming arrows representing state conditions (referring to corresponding instances of other entity types). There are two kinds of outgoing arrows: one for specifying mental effects (changing beliefs and/or commitments) and one for specifying the performance of (physical and communicative) actions. An outgoing arrow with a double arrowhead denotes a mental effect. An outgoing connector to an action event type denotes the performance of actions of that type.

Fig. 2 shows an example of an *interaction pattern diagram* specifying a reaction rule R1 for an elevator in response to the non-action event type `Arrival` at some floor. Notice that all state attributes of the elevator are optional, expressed by the UML multiplicity expression [0..1]. Both the precondition and the postcondition are expressed in UML's Object Constraint Language (OCL). The precondition for halting is `Arrival.Floor = TargetFloor`, the two sequentially triggered action events are `halt` and `closeDoor` and the postcondition representing the mental state effect is `HaltFloor = Arrival.Floor`.

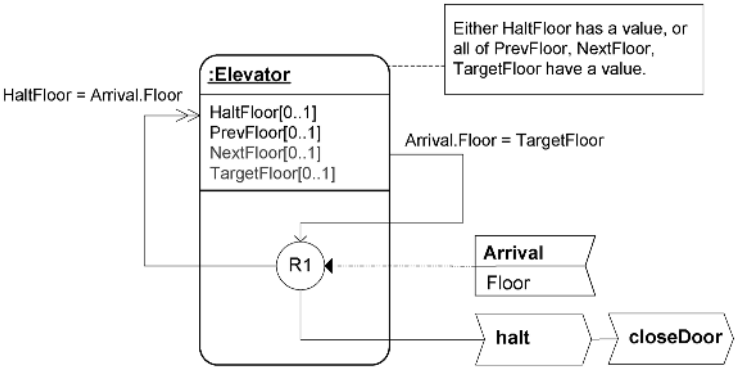

Fig. 2. An interaction pattern: when arriving at a floor, the elevator halts and opens its door.

In symbolic form, a reaction rule is defined as a quadruple

$$\varepsilon, C \rightarrow \alpha, P$$

where ε denotes an event term (the triggering event), C denotes a logical formula (the mental state condition), α denotes an action term (the triggered action) and P denotes a logical formula (specifying the mental effect or postcondition).

2.2 Modelling and Simulating Communication and Perception

For modelling and simulating *communication* between agents, we do not consider non-verbal communication and abstract away from the physical layer, where the speaker realizes a communication act (or, synonymously, sends a message) by performing some physical action (such as making an utterance) and the listener has to perceive this action event, implying that, due to the physical signal transmission, there can be noise in the listener's percepts referring to the message send event. In general, for each (external) *event E* and each agent, the simulation system has to compute a corresponding *potential percept PP* (according to physical laws), from which the *actual percept AP* has to be derived according to the perceptive capability of the agent. The mental availability of the actual percept, then, is the (internal) perception event corresponding to the external event. There are two options for simplifying the $E \rightarrow PP \rightarrow AP$ chain: we can either assume that

1. $E = PP = AP$, so we don't have to distinguish between an external event and the corresponding perception event; or
2. $PP = AP$, that is, all agents have perfect perception capabilities.

For communication events, it makes sense to assume that $E = PP = AP$, i.e. the message received is equal to the corresponding message sent. Yet, there may be a delay between these two events, depending on the type of the message transport channel and the current physical state of the speaker and listener.

For the perception of a non-communicative action event such an assumption may be not justified and would mean a severe simplification. However, the less severe simplification expressed by the assumption that $PP = AP$ may be justified for many purposes.

A *send message* action type is represented internally (in a behaviour rule) in the form of

sM(Receivers, M(C1,C2,...))

where M denotes the message type, and the Ci denote the message content parameters. When such an action is performed (as a consequence of firing the behaviour rule), we obtain external sM event expressions (in the future events list) of the form

sM(EvtID1, Sender, Receivers, M(C1,C2,...)) @ OccTime

corresponding to the external AOR model of a communicative action event (such as in Fig. 6). Such an sM event may be processed by creating *memory facts* of the form

M(EvtID1, OccTime, Sender, Receivers, C1, C2, ...)

and to *receive message* events of the form

rM(EvtID2, Receiver, EvtRef) @ OccTime+D

where the delay D is a non-zero multiple of the cycle duration and EvtRef refers to the EvtID of the corresponding sM event (i.e. EvtRef = EvtID1).

2.3 Examples

We briefly discuss three simple examples.

2.3.1 Example 1: An Elevator

A simple example scenario is an elevator with the action repertoire: moveUp, moveDown, halt, openDoor and closeDoor. The elevator agent has to react to pickup and transport requests from elevator users; when moving, it has to react to events signalling the arrival at a certain floor; and it has to react to timeout signal to close the door. This scenario is described in the AOR diagram in Figure 3.

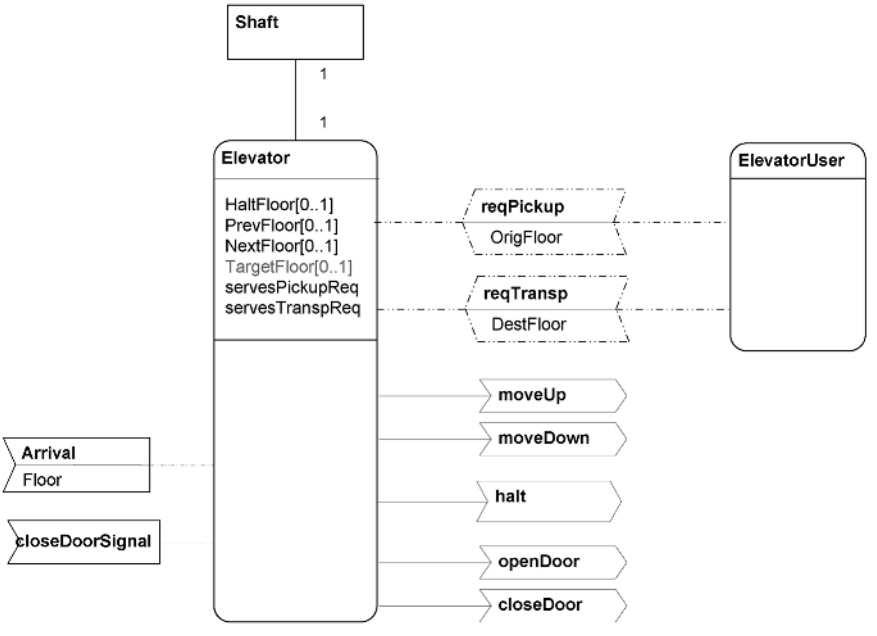

Fig. 3. An elevator scenario as an external AOR model.

An AORS scenario model describes both the systems/agents to be simulated (here the elevator) and the systems/agents outside the simulation borderline they interact with (here the elevator user). In a next step, such a scenario model is transformed into a simulation model in which the external systems are dropped and all action event types in interaction frames involving them (here reqPickup and reqTransp) are turned into corresponding *exogenous event* (here `PickupReq`) or *successor event* types (here `TranspReq`) as shown in Figure 4. ***Exogenous events*** drive the simulation and are generated at random; their periodicity is specified with the help of a tagged value – in the case of PickupReq: `{periodic=exp(…)}`. They are either non-action events that are not caused by other events or external action events in the sense that their actor is not included in the simulation.

In the simulation model of Figure 4, there are also two examples of reaction rules that represent causation rules. In AOR simulation, as explained in section 3, such

rules are used to represent a model of causality that is enacted by the environment simulator. The first causation rule, R1, creates an `Arrival` event, perceived by the elevator with a delay of 7 seconds, upon any `moveUp` or `moveDown` event. The second causation rule, between the `openDoor` action event and the `closeDoorSignal` event, is expressed in a visually simplified notation (an association connection line with stereotype «causes») specifying that, upon opening the door, a specific signal for closing the door is created with a delay of 5 seconds.

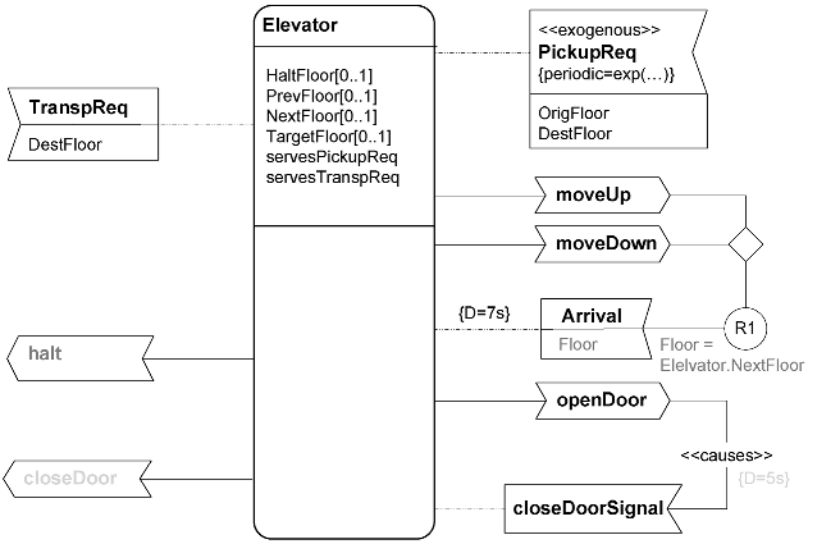

Fig. 4. A simulation model for the elevator system.

2.3.2 Example 2: Communicating Elevators

We consider an example scenario where two elevators operate in the same shaft and must take care to avoid collisions. For simplicity, we restrict our consideration to the case with three floors. Elevator A is serving floor 1 and 2, and elevator B is serving floor 2 and 3, and hence the critical zone, requiring coordination, is floor 2. This scenario is depicted in Fig. 5.

The external AOR diagram in Fig. 6 models this scenario, which requires a model of coordination between the two elevators.

Fig. 5. In addition to the modelling elements used to describe the single elevator scenario, there are two message types (`reqPerm`, `grantPerm`) and a commitment/claim type for `grantPerm` for the coordination between the two elevators.

2.3.3 Example 3: The MIT Beer Game

The MIT Beer Game is a management simulation that was developed by the System Dynamics Group of the Sloan School of Management in the early 1960s.

The game is played by teams of four persons, each person playing one of the four roles of Retailer, Wholesaler, Distributor and Factory. The four roles are arranged in a simple beer supply chain (Fig. 7).

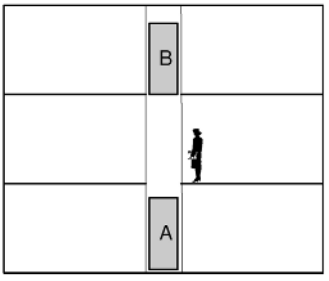

Fig. 5. Two elevators operating in one shaft.

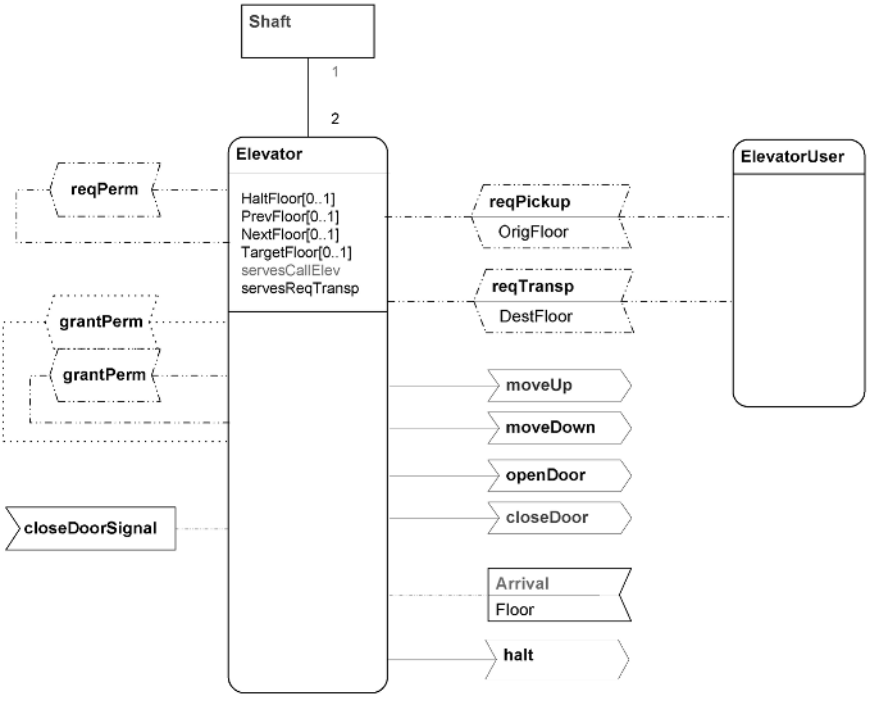

Fig. 6. An external AOR diagram modelling the communicating elevator scenario from.

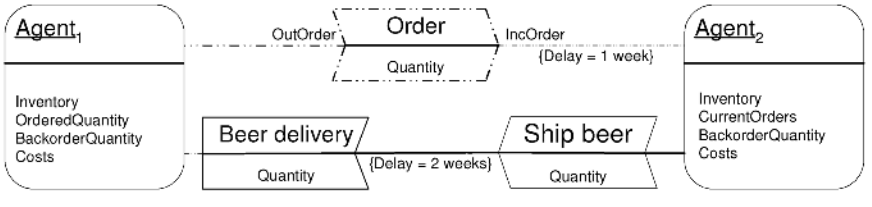

Fig. 7. The interaction frame between any two nodes of the MIT Beer Game supply chain.

The factory has an unlimited supply of raw materials and each of the roles has an unlimited storage capacity. The supply lead times (the time between shipment by the supplier and arrival at its destination) and order delay times (the time between the sending of an order and the arrival of the order at its destination) are fixed.

Supplies take two full turns to arrive; the orders for new beer arrive at their destination in the next turn.

Each turn (usually a simulated week), the retailer receives a customer order and tries to ship the requested amount from its inventory. It then orders an amount of beer from its supplier, the wholesaler, which tries to ship the requested amount from its inventory, and so on. Orders that cannot be met are placed in backorder and must be met as soon as possible.

At the end of each week, each role has to calculate that week's costs. Remaining inventory is charged $0.50 per item as holding costs and backorders are charged $1.00 per item as shortage costs. The objective of the game is to be the team with the lowest overall costs or to be the player with the lowest cost within a team, after playing a fixed number of weeks.

In [LTW04], we present a complete AORS model of the Beer Game.

3 Agent-Based Discrete Event Simulation

In Discrete Event Simulation (DES) systems are modelled in terms of *system states* and *discrete events*, i.e. as *discrete dynamic systems*. Since a system is composed of entities, its state consists of the combination (Cartesian product) of all states of its entities. All state changes are brought about by events.

DES is a very generally applicable and powerful approach, since many systems, in particular technical and social systems, can be viewed as discrete dynamic systems. There are two methods in DES for the advance of simulated time: using *event-based time progression*, the time is advanced according to the occurrence time of the next event; using *incremental time progression*, the simulated time is advanced in regular time steps.

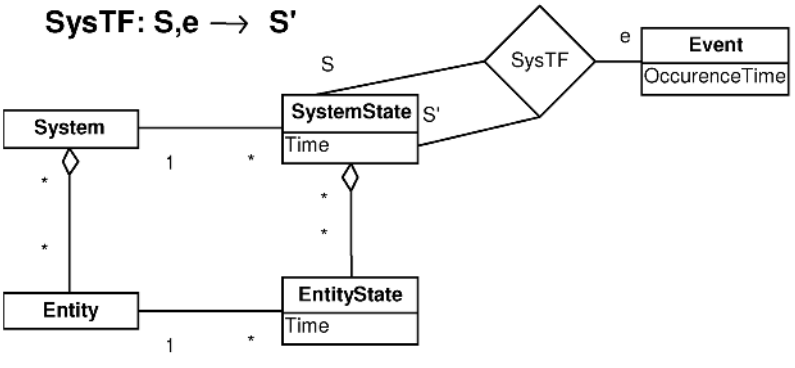

Fig. 8. The basic model of discrete event simulation as a UML class diagram. As a system consists of a number of entities, its state consists of all the states of these entities. A system transition function *SysTF* takes a system state *S* and a state-changing event *e* and determines the resulting successor state *S'*.

In many ABS approaches, the basic DES model is refined in some way by making certain additional conceptual distinctions, including the fundamental distinction between interacting agents and passive objects. These simulation approaches may be classified as *Agent-Based Discrete Event Simulation (ABDES)*.

In our version of ABDES, extending and refining the basic DES model into a model of *Agent-Object-Relationship Simulation (AORS)*, we start with time-driven DES (since we need small regular time steps for simulating the perception-reaction cycle of agents) and adopt a number of essential ontological distinctions from AORML:

- The enduring entities of a system (also called *endurants* in foundational ontologies) are distinguished into *agents* and *objects*.
- Agents maintain *beliefs* (referring to the state of their environment) and process *percepts* (referring to events).
- Events can be either *action events* or *non-action events*.
- Action events can be either *communicative* (messages) or *non-communicative*.

In addition to these conceptual distinctions from AORML, we need to introduce the distinction between *exogenous events* (non-action events that are not caused by other events or external action events in the sense that their actor is not included in the simulation) generated periodically at random and *successor events*, which are either caused events or action events. The basic DES model is shown in Fig. 8 (an extension thereof is depicted later in Fig. 10).

3.1 An Abstract Architecture and Execution Model for ABDES Systems

In ABDES, it is natural to partition the simulation system into

1. the ***environment simulator*** responsible for managing the state of all external (or physical) objects and for the external/physical state of each agent;
2. a number of ***agent simulators*** responsible for managing the internal (or mental) state of agents.

The state of an ABDES system consists of:

- the simulated time *t*
- the environment state representing
  - the non-agentive environment (as a collection of objects) and
  - the external states of all agents (e.g., their physical state, their geographic position etc.)
- the internal agent states (e.g., representing perceptions, beliefs, memory, goals etc.).
- a (possibly empty) list of future events

For each simulation run, a start and an end calendar date-time are specified. The simulated time t represents a cycle counter, which can be transformed into a calendar date-time with the help of a cycle duration constant (for which we use 100 ms by default). The simulation run stops at the end date-time. By default, both the perception-action and the action-perception delay are 100 ms, as illustrated in Fig. 9.

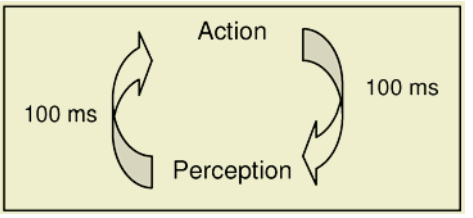

Fig. 9. The perception-action and action-perception delays.

At initialization time, for each exogenous event type, future events are created at random (according to the given distribution function) and put in the future events list.

At runtime, a simulation cycle consists of the following steps:

1. At the beginning of a new simulation cycle, say at simulated time t, the environment simulator determines the current events (all the events of the future events list whose occurrence time is now).
2. The environment simulator determines, on the basis of the current environment state and the current events:
   a) a new environment state
   b) a set of new caused events (with suitable occurrence time) created by the causality rules of the environment simulator, including
      - new caused *environment events*
      - next step *receive message events* (with occurrence time *t+1*) derived from the corresponding current send message events
      - next step *perception events* (with occurrence time *t+1*) derived from the corresponding current *environment events* and *non-communicative action events*
   c) The environment simulator then hands over to each agent simulator the current *receive message events* and *perception events* for the simulated agent.
3. Each agent simulator computes, on the basis of the current internal agent state and the delivered *perception events* and *receive message events*,
   a) a new internal agent state,
   b) the set of new *action events* (with occurrence time *t+1*) created by the behaviour rules of the agent simulator (representing the immediate reactions of the simulated agent in response to the delivered *perceptions* and *message receptions*)
4. The future events list is updated by removing all the processed events and adding the new caused events from step 2b and the new action events from step 3b.
5. If the simulation run continues, the environment simulator sets the simulated time *t*
   a) to *t+1*, if it is in incremental time progression mode, or
   b) to the occurrence time of the next event from the future events list, if it is in event-based time progression mode,

   and starts over with step 1 of the simulation cycle.

The simulation run ends when the future events list is empty or when the end date-time is reached.

This abstract architecture and execution model for ABDES systems can be instantiated by different concrete architectures and systems. In section 4, we present a Prolog program that implements an AORS system and instantiates this architecture.

The AORML distinction between external and internal models provides the means needed for modelling both the environment and the agents involved in a simulation scenario. An external model describes the perspective of the environment simulator, whereas the internal models derived from the external one describe the perspectives of the involved agents. This suggests the following methodology for developing an AOR simulation model:

1. In the domain analysis of the simulation problem, develop an external AOR model of the scenario from the perspective of an external observer. This model is the basis both for designing the environment simulation and for deriving the specification of the involved agent simulators.
2. For each involved agent, transform the external AOR model of the simulation scenario into an internal AOR model for specifying the corresponding agent simulator.

3.2 Advantages of ABDES and AORS

ABDES and AORS support

- structure-preserving modelling and closer-to-reality simulation:
  - Passive entities with certain properties are modelled as objects with corresponding attributes.
  - Interactive entities (actors) are modelled as agents, which have beliefs and perceptions, and interact with each other and with their environment.
- functionally distributed simulation where any of the participating simulators (the environment simulator and all involved agent simulators) may be deployed to different threads or processes, possibly running on different machines (realizing vertical distribution).
- *interactive* simulation where any of the involved agent simulators may be replaced by its real counterpart.
- modelling and simulating *pro-active* behaviour, in addition to the basic reactive behaviour.

4 A Prolog Prototype of an AOR Simulation System

Implemented as a Prolog program, the AOR simulation cycle yields the following procedure:

```
1:  cycle( _, _, _, []) :- !.
2:  cycle( Now, EnvSt, IntAgtSts, EvtList) :-
3:      extractCrtEvts( Now, EvtList, CrtEnvEvts, CrtPercEvts),
4:      envSimulator( Now, CrtEnvEvts, EnvSt, NewEnvSt,
            TranslCausEvts),
5:      agtsSimulator( Now, CrtPercEvts, IntAgtSts,
            NewIntAgtSts, TranslActEvts),
6:      computeNewEvtList( EvtList, CrtEnvEvts, TranslCausEvts,
            TranslActEvts, NewEvtList),
7:      NextMoment is Now+1,
8:      cycle( NextMoment, NewEnvSt, NewIntAgtSts, NewEvtList).
```

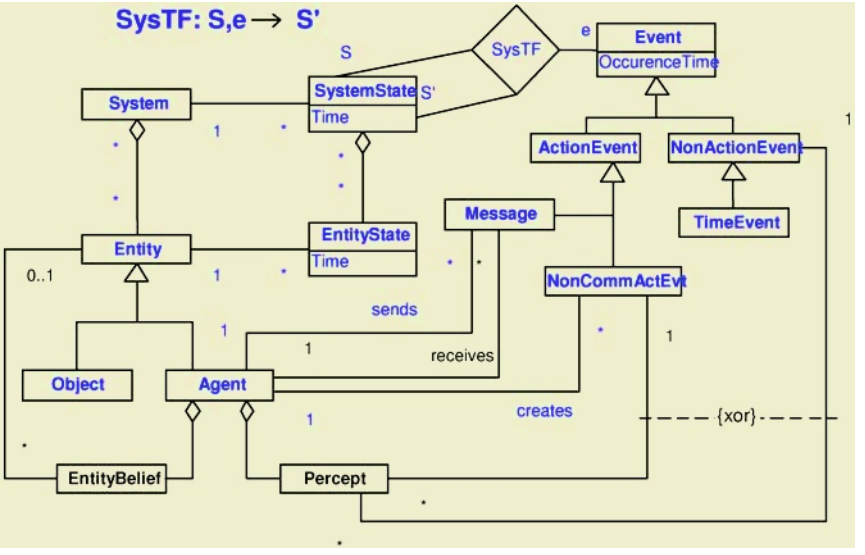

Fig. 10. A UML class diagram describing the basic ontology of AORS: Agents maintain beliefs about entities, send/receive messages, and process percepts, which refer either to a non-communicative action event or to a non-action event. Notice that communication (sending/receiving messages) is separated from perception (perceiving non-communicative action events and non-action events).

Line 1 represents the exit condition (when the future events list is empty). In line 3, the current environment events (steps 1a and 1b of the simulation cycle) and also the current perception events are extracted from the future events list. Lines 4 and 5 simulate the system in the current cycle by first calling the environment simulator and then calling all agents simulators. In line 6, the future events list is updated (step 4). The last two lines update the time and start a new cycle (step 5). `NewEnvSt` and `NewIntAgtSts` stand for the new environment state and the new internal states of agents.

We represent physical causality as a transition function, which takes an environment state and an event and provides a new environment state and a set of caused events. This function is specified as a set of reaction rules for the environment simulator in the form of

```
rrEnv( RuleName, Now, Evt, Cond, CausEvt, Eff)
```

with obvious parameter meanings. Agent behaviour, as a function from a mental state and a perception event to a new mental state and a set of action events, is also specified by a set of reaction rules:

```
rr( AgentName, RuleName, OwnTime, Evt, Cond, ActEvt, Eff)
```

For processing these rules we use two meta-predicates:

1. **`prove`**`( X, P)` where `X` is a list of atomic propositions (representing an environment state or an internal agent state) and `P` is a proposition.
2. **`update`**`( X, P, X')` where `X'` is the new state resulting from updating `X` by assimilating `P` (in our simple example this means asserting/retracting atoms).

When *E* is a current event, and there is an environment simulator rule, whose event term matches *E* such that **prove**(EnvSt, Cond) holds, then the specified CausEvt is added to the caused events list of step 2c) and the environment state is updated by performing

update(EnvSt, Eff, NewEnvSt)

In a similar way, the reaction rules of each agent are applied, updating its internal state by

update(IntAgtSt, Eff, NewIntAgtSt)

We now present the environment simulator:

```
1:   envSimulator( Now, CrtEvts, EnvSt, NewEnvSt, TranslCausEvts) :-
2:       findall( [CausEvt, Eff],
                  (
                  member( Evt/_, CrtEvts),
                  rrEnv( RuleName, Now, Evt, Cond, CausEvt, Eff),
                  prove( EnvSt, Cond)
                  ),
                  ListOfResults),
3:       extractEffects( ListOfResults, Effects),
4:       computeNewEnvState( EnvSt, Effects, NewEnvSt),
5:       extractEvents( ListOfResults, CausEvts),
6:       translateCausEvts( Now, CausEvts, TranslCausEvts).
```

In line 2 all events (and their accompanying effects) that are caused by an event from the CrtEvts list are collected in ListOfResults. Based on the effects of the current environment events (extracted on line 3) the new environment state is determined (line 4). After extracting also the caused events from ListOfResults (in line 5), their absolute time stamp is computed with respect to the current moment (line 6).

A similar procedure is performed for each agent:

```
1:   agtSimulator( AgtName, Now, CrtPercEvts, IntAgtSt,
                   NewIntAgtSt, ActEvts) :-
2:       findall( [ActEvt, Eff],
                  (
                  member( Evt, CrtPercEvents),
                  rr( AgtName, RuleName, OwnTime, Evt, Cond, ActEvt, Eff),
                  prove(IntAgtSt, Cond),
                  ),
                  ListOfResults),
3:       extractEvents( ListOfResults, ActEvts),
4:       extractEffects( ListOfResults, Effects),
5:       computeNewState( IntAgtSt, Effects, NewIntAgtSt).
```

5 Running AORS Models

In AORS, a simulation model is expressed by means of

1. a model of the environment (obtained from the external AOR model of the scenario), consisting of
   - a state structure model specifying all entity types, including exogenous event types
   - a causality model, which is specified by means of reaction rules

2. a model for each involved agent (obtained from internalizing the external AOR model of the scenario into a suitable projection to the mental state of the agent under consideration), consisting of
   - a mental state structure model
   - a behaviour model, which is specified by means of reaction rules
3. a specification of the initial states for the environment and for all agents

The environment and agent models can be defined visually by means of AOR diagrams. Also the initial states can be defined by means of instance diagrams (similar to UML object diagrams). The encoding of a simulation model by means of a high-level UML-based modelling language provides a platform-independent representation and allows the generation of platform-specific code automatically.

In the case of our Prolog simulation platform, we have to generate Prolog predicates from the AOR models. We also have to generate the reaction rules for specifying causality and agent behaviour in the format of the simulator.

Please consult the web page

`http://AORS.rezearch.info`

for obtaining further information about AORS and for downloading our Prolog AORS system.

6 Related Work

Agent-Based Simulation is being used in various research areas, today. In particular, it is being used in

- ***Biology***, e.g. for investigating eco-systems or in population ethology (especially with respect to ants and other insects), see, e.g., [Klü01];
- ***Engineering*****:** for analysing and designing complex (socio-) technical systems, such as Automatically Guided Vehicle Transport Systems [RW02];
- ***Economics*****:** e.g. in the simulation of auctions and markets, see *Trading Agent Competition* [TAC02];
- ***Social Sciences*****:** e.g. in [CD01] the phenomena of social monitoring and norm-based social influence and in [Hal02] the cooperation in teams is studied.

Some well-known platforms for Agent-Based Simulation are *Swarm* [Swarm00, Swarm96], SDML [MGWE98], *Sesam* [Klü01], *MadKit* [MadKit00] and *CORMAS* [Cormas01, Cormas00]. A particularly interesting class of simulation systems is formed by international technology competitions such as *RoboCup* [Robo98] and *Trading Agent Competition (TAC)* [TAC02]. Both RoboCup and TAC can be classified as interactive agent-based realtime simulation systems.

Although there is a large body of work on agent-based simulation, our AORS approach appears to be the first general UML-based declarative approach to agent-based discrete event simulation.

7 Conclusions

We have presented a general approach to modelling and simulating scenarios of interacting systems as multiagent systems, based on the *Agent-Object-Relationship (AOR)*

modelling language. Our Prolog implementation of an AOR simulation system is still in an early prototype stage. In the future, we will transfer it to the Java platform.

References

[Boo99] G. Booth: CourseWare Programmer's Guide, Yale Institute for Biospheric Studies, 1999.

[Cormas00] C. Le Page, F. Bousquet, I. Bakam, A. Bah, C. Baron: CORMAS: A multiagent simulation toolkit to model natural and social dynamics at multiple scales. In Proceedings of Workshop "The ecology of scales", Wageningen (The Netherlands), 2000.

[Cormas01] CIRAD: CORMAS, Common-pool Resources and Multi-Agents Systems, User's Guide, 2001.

[CD01] Rosaria Conte and Frank Dignum. From Social Monitoring to Normative Influence. *Journal of Artificial Societies and Social Simulation* 4:2 (2001).

[Dav02] Paul Davidsson. Agent Based Social Simulation: A Computer Science View. Jo*urnal of Artificial Societies and Social Simulation*, 5:1 (2002).

[Dav02] A. Davies: EcoSim: An Interactive Simulation, Duquesne Universität, Pittsburgh, 2002.

[Den71] D.C. Dennett: Intentional Systems. *The Journal of Philosophy*, 68 (1971).

[EW02] B. Edmonds, S. Wallis: Towards an Ideal Social Simulation Language, Manchester Metropolitan University, 2002.

[FG98] J. Ferber, O. Gutknecht: A meta-model for the analysis and design of organizations in multi-agent systems. Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS´98), IEEE Computer Society Press, pp. 128–135, 1998.

[Hal02] D. Hales: Evolving Specialisation, Altruism and Group-Level Optimisation Using Tags. Presented to the MABS'02 workshop at the AAMAS 2002 Conference. Springer-Verlag, LNCS, 2002.

[HLA02]: Defense Modelling and Simulation Office: High Level Architecture, 2002.

[Jac94] I. Jacobson. *The Object Advantage*. Addison-Wesley, Workingham (England), 1994.

[Klü01] F. Klügl: *Multiagentensimulation*, Addison-Wesley Verlag, 2001.

[LTW04] J.v.Luin, F. Tulba, G. Wagner: Remodelling The Beer Game As An
Agent-Object-Relationship Simulation, submitted to ABS 5.

[MadKit00] J. Ferber, O. Gutknecht, F. Michel: MadKit Development Guide, 2002.

[MGWE98] Moss, Scott , Helen Gaylard, Steve Wallis and Bruce Edmonds, SDML: A Multi-Agent Language for Organizational Modelling, *Computational and MathematicalOrganization Theory* 4:1 (1998), 43–70.

[Robo98] I. Noda, H. Matsubara, K. Hiraki, I. Frank: Soccer Server: a tool for research on multiagent systems, *Applied Artificial Intelligence*, 12:2-3 (1998).

[MBLA96] N. Minar, R. Burkhart, C. Langton, M. Askenazi: The Swarm Simulation System: A Toolkit For Building Multi-Agent Simulations, 1996.

[Swarm00] P. Johnson, A. Lancaster: Swarm User Guide, 2000.

[TAC02] SICS: Trading Agent Competition 2002. See http://www.sics.se/tac/.

[Wag03] G. Wagner. The Agent-Object-Relationship Meta-Model: Towards a Unified View of State and Behaviour, *Information Systems* 28:5 (2003), 475–504.

[WT03] G. Wagner and F. Tulba: Agent-Oriented Modelling and Agent-Based Simulation. In Jeusfeld, M.A. and Pastor, O. (Eds.), *Conceptual Modelling for Novel Application Domains*, volume 2814 of Lecture Notes in Computer Science, Springer-Verlag, pp. 205-216, 2003.

Modelling Institutional, Communicative and Physical Domains in Agent Oriented Information Systems

Maria Bergholtz, Prasad Jayaweera, Paul Johannesson, and Petia Wohed

Department of Computer and Systems Sciences
Stockholm University and The Royal Institute of Technology
Forum 100, SE-164 40 Kista, Sweden
{maria,prasad,pajo,petia}@dsv.su.se

Abstract. One role of a business system is to provide a representation of a Universe of Discourse, which reflects its structure and behaviour. An equally important function of the system is to support communication within an organisation, by structuring and co-ordinating the actions performed by the organisation's agents. These two roles of a business system may be represented in terms of business and process models, i.e. separating the declarative aspects from the procedural control flow aspects of the system. Although this separation of concerns has many advantages, the differences in representation techniques and focus of the two model types constitute a problem in itself. Abstracting business semantics out of, for instance, technical messaging protocols poses severe problems for business analysts. The main contribution of this paper is a unified framework based on agent oriented concepts for facilitating analysis and integration of business models and process models in a systematic way. The approach suggested bridges the gap between the declarative and social/ economic aspects of a business model and the procedural and communicative aspects of a process model in a technology independent manner. We illustrate how our approach can simplify business and process models integration, process specification, process pattern interpretation and process choreography.

1 Introduction

Agent oriented concepts, like agents, events, actions, commitments etc, have recently been introduced in the area of information systems analysis and design, thereby extending the conceptual toolkit for object-oriented analysis and design. The driving force for this has been the need of adequate capture and representation of the semantics of business processes. Two main points, when arguing for this [22], have been that i) the state of an agent also includes mental components, such as beliefs and commitments, which are not captured by the existing conceptual frameworks available within object-oriented analysis, and ii) the communication between agents is realised through speech acts, which are application independent in contrast to the application specific, ad-hoc manner within the object-oriented paradigm.

Furthermore, addressing the business process analysis for e-Commerce, a distinction between business models and process models has been made [11]. A business model is concerned with value exchanges among business partners [11], while a process model focuses on operational and procedural aspects of business communication.

P. Giorgini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 189–205, 2004.

This means that the process of designing e-Commerce systems consists of two main phases: firstly, a business requirement capture phase focusing on value exchanges, and secondly, a phase focused on operational and procedural realisation.

In the business requirement capture phase, coarse-grained views of business activities as well as their relationships and arrangements in business collaborations are represented by means of business model constructs at an abstract level. In contrast, the specification of a process model deals with more fine-grained views of business transactions, their relationships and choreography in business collaborations. Although the two phases in e-Commerce design, and their related models, have different foci, there is clearly a need for integrating them. A unified framework covering coarse-grained business modelling views to fine-grained process specification views provides several benefits. It can be used for supporting different user views of the system being designed and it can form the basis of a precise understanding of modelling views and their inter-relationships. It can also provide a basis for design guidelines that can assist in developing process models.

The purpose of this paper is to propose a unified framework integrating the contents of business models and process models. The framework is based on agent-oriented concepts, like agent, commitment, event, action, etc., [20]. We use ebXML [9] and UMM [2] as the basis of our framework, more specifically the UMM Business Requirements View (BRV) for business models and the UMM Business Transaction View (BTV) for process models. UMM BRV already includes a number of agent-oriented concepts, which we extend by adding a number of constructs for bridging business and process models, in particular speech acts. The work presented in this paper builds on [6] and [5], where speech act theory [19] and the language/action approach [7], [21] are used for analysing processes, as well as for clarifying the relationships between agents in business and process models, or for coordinating web services.

The rest of the paper is organised as follows. Section 2 gives an overview of related research and introduces informally the basic concepts. Section 3 introduces the UMM BRV and BTV. Section 4 contains the main contribution of the paper and presents the integrated framework. Section 5 illustrates two applications of the introduced framework, together with the analysis and design of business process patterns. Section 6 introduces rules for governing the choreography of transactions and collaborations. Section 7, finally, concludes the paper and discusses the results.

2 Basic Concepts and Related Research

A starting point for understanding the relationships between business models and process models is the observation that a person can carry out several different actions by performing a single physical act. An everyday example could be a person who turns on the water sprinkler and thereby both waters the lawn and fulfils the promise to take care of the garden – one physical act (turning on the sprinkler), which can be viewed as "carrying" two other actions (watering the lawn and fulfilling a promise). Relationships like these are particularly common for communicative actions, which are carried out by means of physical actions. One way to look at the role of communicative actions and their relationships to other actions is to view human actions as taking place in three different domains:

* ***The physical domain.*** In this domain, people carry out physical actions – they utter sounds, wave their hands, send electronic messages etc.
* ***The communicative domain***. In this domain, people express their intentions and feelings. They tell other people what they know and they try to influence the behaviour of other actors by communicating with them. People perform such communicative actions by performing actions in the physical domain.
* ***The social/institutional domain.*** In this domain, people change the social and institutional relationships among them. For example, people become married or they acquire property. People change social and institutional relationships by performing actions in the communicative domain.

Using this division, business models can be seen as describing the social/institutional domain, in particular economic relationships and actions like ownership and resource transfers. Process models, on the other hand, describe the communicative domain, in particular how people establish and fulfil obligations. A similar approach is shown in [13] where a set of social patterns to bridge the gap between program driven and requirements driven paradigms to information systems modelling is introduced. Our work targets this work and aims at creating a set of concepts to unify already existing models in e-Commerce systems development.

The three-fold division above is based on an agent-oriented approach to information systems design [22]. A key assumption of this approach is that an enterprise can be viewed as a set of co-operating agents that establish, modify, cancel and fulfil commitments and contracts [8]. In carrying out these activities, agents rely on so called speech acts, which are actions that change the universe of discourse when a speaker utters them and a recipient grasps them. A speech act may be oral as well as written, or even expressed via some other communication form such as sign language.

The feasibility of speech act theory for electronic communication systems is supported by several researchers, see [18] for a review. The work reported on in this paper differs from these approaches since it uses speech act theory for analysing and integrating different modelling domains in e-Commerce, rather than facilitating electronic message handling per se.

One of the pioneers in the development of a theory of speech acts is John Searle, [19], who introduced a taxonomy of five different kinds of speech acts: assertive, directive, commissive, expressive and declarative, also called illocutionary points.

An *assertive* is a speech act the purpose of which is to convey information about some state of affairs of the world from one agent, the speaker, to another, the hearer. A *commissive* is a speech act, the purpose of which is to commit the speaker to carry out some action or to bring about some state of affairs. A *directive* is a speech act, where the speaker requests the hearer to carry out some action or to bring about some state of affairs. A *declarative* is a speech act where the speaker brings about some state of affairs by the mere performance of the speech act, e.g. “I declare you husband and wife”. Finally, an *expressive* is a speech act, the purpose of which is to express the speaker’s attitude to some state of affairs.

In addition to its illocutionary point, a speech act also has a propositional content. The speech acts “I hereby pronounce you husband and wife” and “You are hereby divorced”, which are both declaratives, have different propositional contents. A speech act is often viewed as consisting of two parts: its propositional content and its illocutionary force. The illocutionary force is the illocutionary point together with the

manner (for example ordering, asking, begging) in which the speech act is performed and the context in which it occurs.

3 UMM Business and Process Models – BRV and BTV

The Resource-Event-Agent (REA) [17] framework has recently been applied in the UN/CEFACT Modelling Methodology (UMM) for business process modelling [2]. The scope of UMM is to provide a procedure for specifying, in a technology-neutral and implementation-independent manner, business processes involving information exchange. In UMM, a number of meta-models are defined to support an incremental model development and to provide different levels of specification granularity.

- A business meta-model, called the *Business Operations Map* (BOM), partitions business processes into business areas and business categories.
- A requirements meta-model, called the *Business Requirements View* (BRV) specifies business processes and business collaborations.
- An analysis meta-model, called the *Business Transaction View* (BTV), captures the semantics of business information entities and their flow of exchange between business partners as they perform business activities.
- A design meta-model, called the *Business Service View* (BSV), models the network components services and agents and their message exchange.

The two meta-models relevant for our work are BRV and BTV (see Fig. 1) and we describe them briefly in the following sub sections.

3.1 Business Requirements View

As it is based on REA, BRV models EconomicEvents, the Resources transferred through the EconomicEvents and the Agents, here called Partners between whom the Economic Events are performed. An EconomicEvent is the transfer of control of a Resource from one Partner to another. Each EconomicEvent has a counterpart, i.e. another EconomicEvent that is performed in return and realising an exchange. For instance, the counterpart of a goods transfer economic event could be a payment, i.e. a transfer of money economic event. This connection between two economic events is modelled through the relationship *duality*. Furthermore, an EconomicEvent fulfils an Economic Commitment. An EconomicCommitment can be seen as the result of a commissive speech act and is intended to model an obligation for the performance of an Economic Event. The *duality* between EconomicEvents is inherited into the Economic Commitments, where it is represented by the relationship *reciprocal*.

In order to represent collections of related commitments, the concept of Economic Contract is used. An EconomicContract is an aggregation of two or more reciprocal Economic Commitments. An example of an EconomicContract is a purchase order composed of one or more order lines, each one representing a corresponding EconomicCommitment in the contract. The product type specified in each line is the EconomicResourceType that is the subject for the EconomicCommitment. EconomicContracts are often made within the boundaries of different Agreements. An Agreement is an arrangement between two Partners that specifies the conditions under which they will trade.

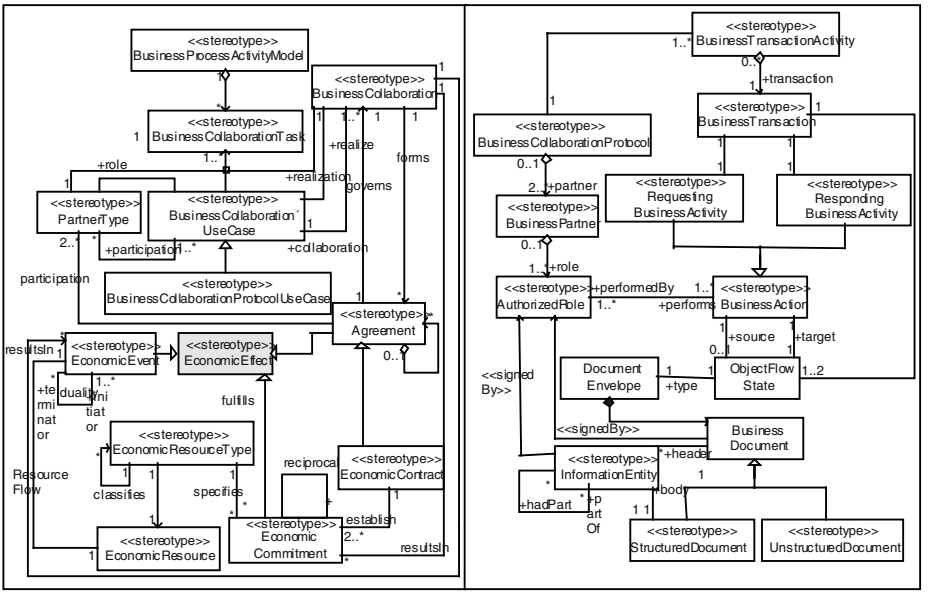

Fig. 1. UMM Business Requirement (BRV)[1] and Business Transaction Views (BTV).

3.2 Business Transaction View

The Business Transaction View (BTV) specifies the flow of business information between business roles as they perform business activities. A BusinessTransaction is a unit of work through which information and signals are exchanged (in agreed format, sequence and time interval) between two business partners. These information exchange chunks, called BusinessActions, are either RequestingBusinessActivities or RespondingBusinessActivities (depending on whether they are performed by a AuthorizedRole who is requesting a business service or whether they are the response to such a request). Furthermore, the flow between different BusinessTransactions can be choreographed through BusinessCollaborationProtocols.

4 An Agent-Oriented Integration Framework

In terms of the three domains introduced in Section 2, UMM explicitly addresses only the physical and the social/institutional domains. The physical domain is modelled through classes like BusinessTransaction and BusinessAction, while the social/ institutional domain is modelled through EconomicCommitment, EconomicEvent, and other classes. The details of the communicative domain, however, are not explicitly modelled. This state of affairs causes two main problems. First, the relationship between the physical and the social/institutional domains is very coarsely modelled; essentially the UMM only states that a completed collaboration may influence objects

[1] EconomicEffect is an extension to UMM BRV and is described in the next section.

in the social/institutional domain, but it does not say how the components of a collaboration affect the social/institutional objects. Secondly, there is no structured or systematic way of specifying how events in the physical domain influence the social/institutional domain. These problems can be overcome by introducing the communicative domain as an additional layer in the UMM, thereby creating a bridge between the physical and social/institutional domains.

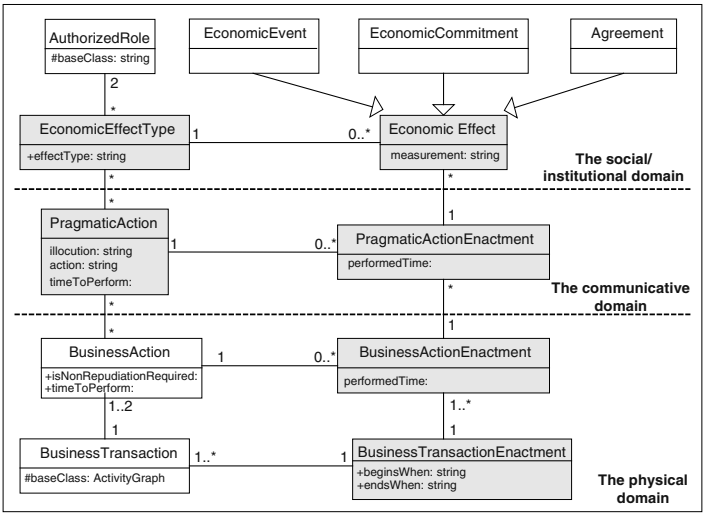

Fig. 2. Extended Business Requirement View.

As a preparation to modelling the communicative domain, a minor modification to UMM BRV is made, see Fig. 2. A class EconomicEffect is introduced as a superclass of EconomicCommitment, Agreement, and EconomicEvent.

The power type [16] of EconomicEffect, called EconomicEffectType, is also added for the purpose of differentiating between the modelling of concrete, tangible objects in a domain, and the abstract characteristic categories of these objects.

These modifications will allow for a more concise representation of the effects of communicative actions. In addition to these changes, the classes BusinessActionEnactment and BusinessTransactionEnactment are added. These represent the actual execution of a business action or business transaction, respectively.

The basic notions introduced for modelling the communicative domain are those of a pragmatic action and its execution, i.e. PragmaticAction and PragmaticActionEnactment, see Fig. 2. A *pragmatic action* is a speech act as introduced in Section 2. It consists of three parts, denoted as a triple:

<Illocution, Action, EffectType>

Intuitively, these components of a pragmatic action mean the following:

- EffectType specifies an EconomicEffectType, i.e. it tells what kind of object the pragmatic action may affect
- Action is the type of action to be applied – create, change, or cancel
- Illocution specifies the illocutionary force of the pragmatic action, i.e. it says what intention the actor has to the Action on the EffectType

Formally, Illocution and Action are defined through enumeration:

Action ∈ {create, change, cancel, none}

Illocution ∈ {propose, accept, reject, declare, query, reply, assert}

The meanings of the illocutions are as follows:

propose – someone proposes to create, change, or cancel an object

accept – someone accepts a previous proposal

reject – someone rejects a previous proposal

declare – someone unilaterally creates, changes, or cancels an object

query – someone asks for information

reply – someone replies to a previous query

assert – someone makes a statement about one or several objects

For 'query', 'reply', and 'assert', there is no relevant Action involved, so only the "dummy" 'none' can be used.

The class PragmaticActionEnactment is used to represent the actual executions of pragmatic actions. A PragmaticActionEnactment specifies a PragmaticAction as well as an EconomicEffect, i.e. the agreement, commitment or economic event to be affected. Some examples of PragmaticActions are:

"Query status of a sales order" would be modelled as <query, none, salesOrder>

"Request purchase order" would be modelled as <propose, create, purchaseOrder>, where 'salesOrder' and 'purchaseOrder' are EconomicEffectTypes

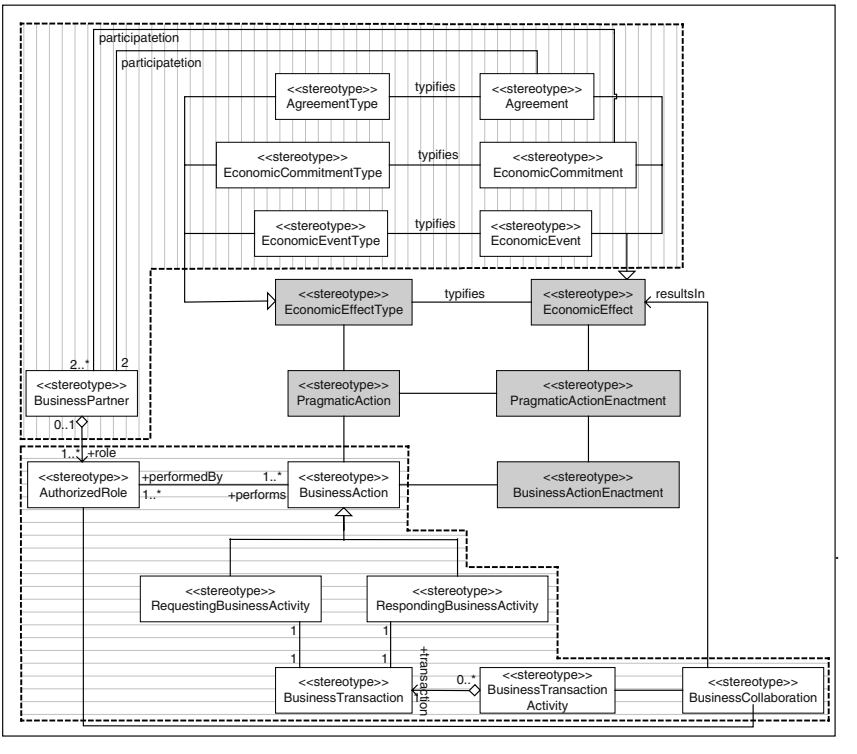

Fig. 3. Integrated Global view.

4.1 Integrated View of Process and Business Models

The glue between the physical domain and the communicative domain is made up by the associations between the classes BusinessAction and PragmaticAction, and BusinessActionEnactment and PragmaticActionEnactment. These associations express that a business action can carry one or more pragmatic actions, i.e. by performing a business action, an actor simultaneously performs one or several pragmatic actions. Often, only one pragmatic action is performed, but in some cases several can be performed, e.g. when creating a commitment and its contract at the same time.

The global integrated view of BRV and BTV is shown graphically in Fig. 3. The original BTV-parts are grouped within the area shadowed with horizontal lines, BRV-parts are grouped within the area shadowed with vertical lines and the new parts introduced in this chapter are depicted in the white area.

5 Application/Analysis of Transaction and Collaboration Patterns

In this section, a number of applications of the proposed framework with respect to business modelling patterns are introduced. A *pattern* is a description of a problem, its solution, when to apply the solution, and when and how to apply the solution in new contexts [14]. Firstly, we discuss how the framework can be used for analysing the semantics of UMM business transaction patterns. Secondly, different collaboration patterns for incremental development are suggested.

5.1 Analysing UMM Business Transaction Patterns

UN/CEFACT has defined a number of business transaction patterns as part of UMM with the intention of providing an established semantics of frequently occurring business interactions. Below, we list a number of these patterns and show how they can be understood based on the framework introduced in the previous section.

Design patterns are defined as "descriptions of communicating objects and classes customised to solve a general design problem in a particular context" [10]. We will adopt this definition to the UMM transaction patterns and view a *transaction pattern* as a template of exactly one pair of a Requesting and Responding Business Activity customised to encode the intentions and effects of a business interaction in a context.

Definition: A *transaction pattern* (TP) is an activity diagram with two states designating the Requesting and Responding Business Activity. Every other state is either the start state or an end state. A state transition from a Requesting or Responding Business Activity is labelled by the pragmatic action(s) carried by the activity, see Fig. 4, Fig. 5 and Table 1, Table 2 below.

The analysis suggests one way to interpret the definitions of the UMM transaction patterns but it does not make any claims to be the final, "correct" interpretation of these definitions. This is not an achievable goal as the definitions are only formulated in natural language, sometimes quite vaguely. The value of the analysis is that it provides explicit interpretations that can be judged for their validity, and thereby can help in formulating more precise and unambiguous patterns.

Table 1. Analysis of UMM transaction patterns in terms of pragmatic actions.

TP	Definition	Analysis
Commercial (Offer/ Accept)	"This design pattern is best used to model the '*offer and acceptance*' business transaction process that results in a residual obligation between both parties to fulfil the terms of the contract. The pattern specifies an originating business activity sending a business document to a responding business activity that may return a business signal or business document as the last responding message." [2]	**Request** <Propose,Create,aContract> **Response** <Accept,Create,aContract> or <Reject,Create,aContract>
Query/ Response	"The query/response design pattern specifies a query for information that a responding partner already has e.g. against a fixed data set that resides in a database. The response comprises zero or more results each of which meets the constraining criterion in the query." [2]	**Request** <Query,None,anEffectType> **Response** <Reply,None,anEffectType>or <Reject,None,anEffectType>
Request/ Confirm	"The request/confirm activity pattern shall be used for business contracts when an initiating partner requests confirmation about their status with respect to previously established contracts or with respect to a responding partner's business rules." [2]	**Request** <Query,None,aCommittment/ aContract> **Response** <Reply,None,aCommitment/ aContract>
Request/ Response	"The request/response activity pattern shall be used for business contracts when an initiating partner requests information that a responding partner already has and when the request for business information requires a complex interdependent set of results." [2]	**Request** <Query, None, anEffectType>or <Reject,None,anEffectType> **Response** <Reply,None,anEffectType>[2]
Information Distribution	"This pattern specifies the exchange of a requesting business document and the return of an acknowledgement of receipt signal. The pattern is used to model an *informal* information exchange business transaction that therefore has no nonrepudiation requirements." [2]	**Request** <Assert,None,anEffectType> **Response** Carries no pragmatic action
Notification	"This pattern specifies the exchange of a requesting business document and the return of an acknowledgement of receipt signal. The pattern is used to model a *formal* information exchange business transaction that therefore has nonrepudiation requirements." [2]	**Request** <Declare,Create,aCommitment/aContract> **Response** Carries no pragmatic action [3].

[2] Note that the analysis fails to make a distinction between the query/response and the request/response patterns; the reason for this is that the difference between the patterns does not reside in different business effects but in different ways of computing the responses.

[3] The motivation for this analysis is that a notification results in a binding specification of business conditions for the initiating partner and, thus, in a (partial) agreement.

Another use of the analysis is to suggest additional patterns than those already present in UMM. The Fulfilment, ContractProposal, Bilateral and Unilateral Cancellations (see Table 2) are obvious candidates.

Table 2. Additional Transaction Patterns to UMM.

TP	Definition	Analysis
Fulfilment	The fulfilment pattern specifies the completion of an Economic Event. Fig. 4	**Request** <Propose,Create,anEconomicEvent>
		Response <Accept,Create,anEconomicEvent>or <Reject,Create,anEconomicEvent>
Contract Proposal	The Contract Proposal Transaction Pattern is a variation of the aforementioned Offer-Accept transaction pattern where the Partners does not have to make their assertions of intentions legally binding.Fig. 4	**Request** <Propose,None,anEconomicContract>
		Response <Accept,None,anEconomicContract>or <Reject,None,anEconomicContract>
Bilateral Cancellation	The Bilateral Cancellation transaction pattern refers to the bilateral cancellation of an Economic Contract or to Commitment(s) within an Economic Contract. See the left part of Fig. 5	**Request** <Propose,Cancel,aContract/Commitment>
		Response <Accept,Cancel,aContract/Commitmntor <Reject,Cancel,aContract/Commitment>
Unilateral Cancellation	The Unilateral Cancellation transaction pattern refers to the unilateral cancellation of an Economic Contract or to Commitment(s) within an Economic Contract. See the right part of Fig. 5	**Request** <Declare,Cancel,aContract/Commitment>
		Response Carries no pragmatic action

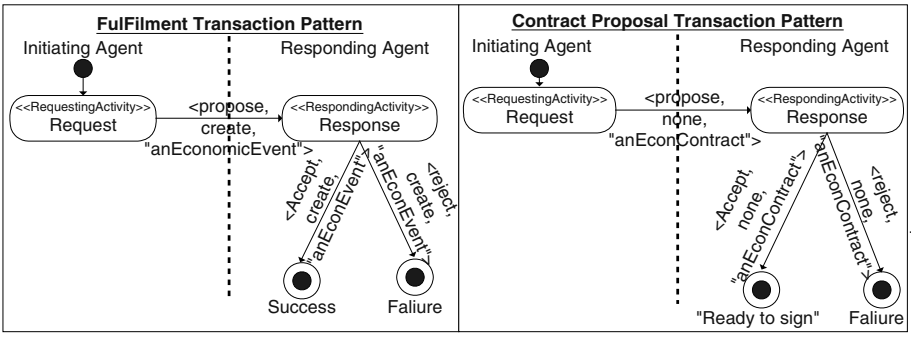

Fig. 4. Fulfilment and Contract Proposal Transaction Patterns.

5.2 Collaboration Patterns

A *Business Collaboration Pattern* defines the orchestration of activities between partners by defining a set of BusinessTransaction patterns and/or more basic collaboration patterns plus the rules for transitioning from one transaction/collaboration to another [1]. The significance of a Business Collaboration Pattern is to serve as a predefined template in that it encodes business rules and business structure according to well-established best practices.

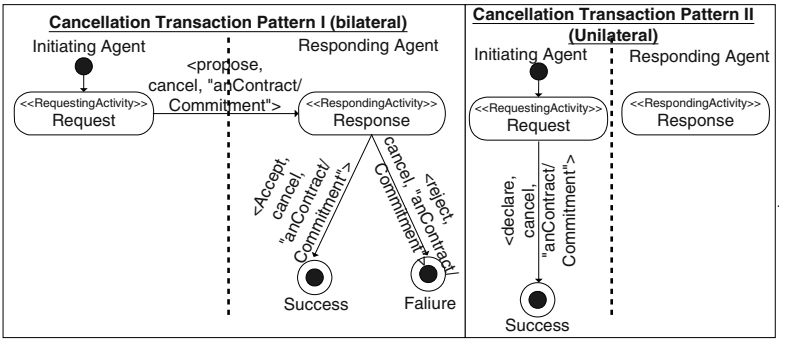

Fig. 5. Bilateral and Unilateral Cancellation Transaction Patterns.

A problem with the UMM collaboration patterns is that their complexity increases dramatically as new patterns are assembled from basic patterns, making the resulting activity diagrams hard to understand. To overcome this difficulty we use a layered approach where the transaction patterns constitute nodes in the activity diagram of a collaboration pattern. In this way, the interactions between business partners within a transaction are modelled in a set of well-defined transaction patterns. In the collaboration pattern, this complexity is hidden and only the outcome of the transaction pattern is taken into consideration.

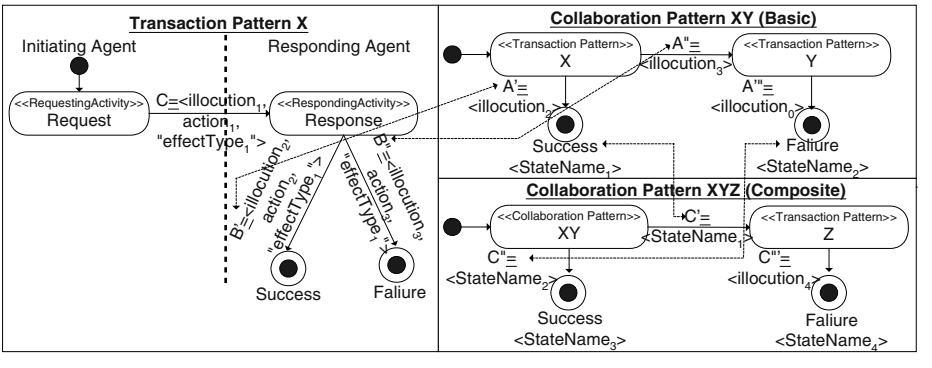

Fig. 6. Generic Transaction Pattern, Basic and Composite Collaboration Patterns.

Definition: A *collaboration pattern* is a state chart over transaction and collaboration pattern(s). A collaboration pattern has exactly two end states representing success or failure of the collaboration, respectively. A transition *A* (see A´/A´´ of Fig. 6) from a transaction pattern must correspond to a transition *B* (see B´/B´´ of Fig. 6) to an end state in that transaction pattern. Furthermore, *A* is labelled (see illocution$_2$/illocution$_3$ of Collaboration Pattern XY of Fig. 6) by the illocution of *B*. A transition *C* (see C´/C´´ of Fig. 6) from a collaboration pattern is labelled by the names of the end states (see StateName$_1$/StateName$_2$ of Collaboration Pattern XYZ of Fig. 6) of the corresponding collaboration pattern.

5.2.1 Fulfilment Collaboration Pattern

The Fulfilment collaboration pattern specifies relevant transaction patterns (see the rightmost part of Fig. 7) and the rules for transitioning among these within the completion of an EconomicEvent. The pattern is assembled from the Fulfilment and Unilateral Cancellation transaction patterns defined in the previous section.

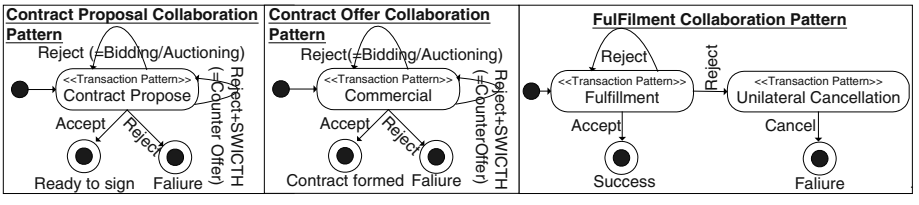

Fig. 7. Contract Proposal, Contract Offer and Fulfilment Collaboration Patterns.

5.2.2 Contract Proposal and Contract Offer Collaboration Patterns

Two basic collaboration patterns for business negotiation for contract formation are given in the Proposal and Offer collaboration patterns [2]. The Contract Proposal collaboration pattern models the non-legally binding negotiation phase in a contract formation, whereas the Contract Offer collaboration pattern expresses the formal creation phase of a contract, see the leftmost and the middle parts of Fig. 7. These patterns are assembled from the Contract proposal transaction pattern and Commercial transaction pattern (described in Section 5.1), respectively.

The two recursive paths when a contract offer/proposal has been rejected have a natural correspondence in the business concepts 'Counter Offer' and 'Bidding' (or 'Auctioning'), respectively. 'Counter Offer' models the situation when the responding agent has rejected the requesting agent's offer and makes a new offer of her own. (The label 'SWITCH' indicates that the roles of the agents are reversed – the agent who received an offer becomes the one who proposes a new offer.) 'Bidding' models the situation where the responding agent has rejected the requesting agent's contract offer and the latter then initiates a new Transaction with a new (changed) offer.

5.2.3 Composite Collaborations

More complex modelling and assembly of commitments, contracts and fulfilments are expressed in the example patterns found in [1]: a) Business Negotiation pattern, b) Order-Fulfilment-Settlement pattern, c) Long-term contract pattern with periodic

releases, d) Escalating commitments pattern, e) Customer order direct delivery pattern.

We will apply our framework for analysing the first collaboration pattern in the list above: the Business Negotiation pattern. This pattern is composed of, in turn, the previously defined transaction patterns: Query-Response transaction pattern, Contract Proposal and Commercial transaction patterns, see the left part of Fig. 8.

An example of a more complex collaboration pattern, composed of basic collaboration patterns only, is also given below. In this example, the Business Negotiation pattern is a part of more complex collaboration patterns, see the right part of Fig. 8 where we envisage a larger collaboration pattern named Contract Fulfillment. As can be seen, the collaboration pattern is formed of a number of collaboration patterns where the two base patterns that make up are the Business Negotiation collaboration pattern and the Fulfilment collaboration pattern.

The three basic collaborations patterns in composite Business Negotiation and Fulfillment Collaboration in Fig. 8 can easily be mapped into typical phases in eContracting process; for instance, those introduced in [4] namely, Information Phase, Pre-Contracting Phase, Contracting Phase and finally Enactment Phase.

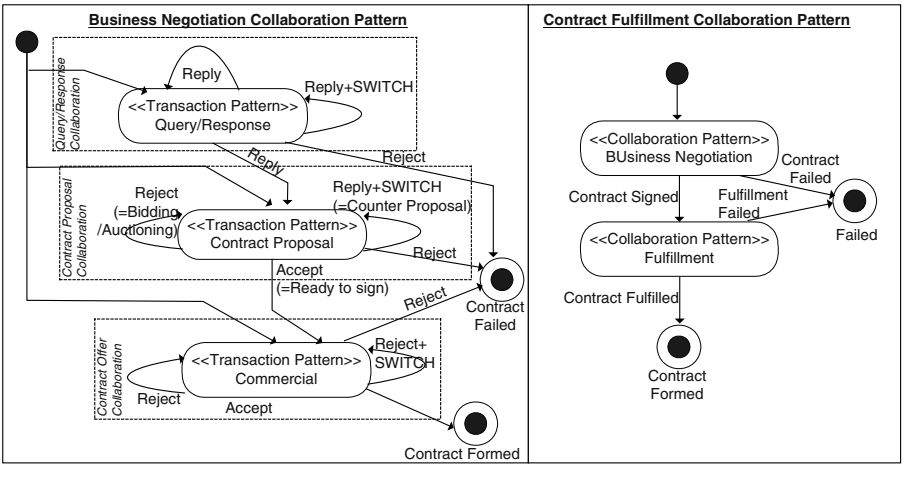

Fig. 8. Business Negotiation and Contract Fulfillment Collaboration Pattern.

6 Choreography of Transactions and Collaborations

Patterns evolve from structures and/or interactions that occur frequently in a certain context or domain. An issue is how to combine the patterns, i.e. how to avoid combining them in an incorrect way that diminishes their usefulness in solving problems. In this section, we propose rules governing the choreography, i.e. the sequencing of business transactions and business collaborations.

6.1 Ordering of Transactions

When a designer constructs a choreography for a collaboration, it is helpful to consider the dependencies that exist among the transactions of the collaboration. Two kinds of dependencies occur across many domains: trust dependencies [12] and flow dependencies [15].

A *trust dependency* is an ordered pair of transactions <A, B> that expresses that A has to be performed before B as a consequence of limited trust between the initiator and the responder. As an example, it is possible to require that a product be paid before it can be delivered.

A *flow dependency* is an ordered pair of transactions <A, B> that expresses that A has to be performed before B because the Economic Resources obtained in A are needed for carrying out B.

We now define two partial orders, Flow and Trust, whose members are ordered pairs of BusinessTransactions between whom a trust or flow dependency holds. Furthermore, a BusinessTransaction is classified according to the Economic EffectType of the pragmatic action it targets, i.e. the EconomicContract, EconomicCommitment, or EconomicEvent to be affected. Fulfilment transactions targets EconomicEvents, commitment transactions targets EconomicCommitments and contract transactions target EconomicContracts. Cancellation transactions refer to all types of pragmatic actions, where the Action is of type 'Cancel'. The signatures of the partial orders are given below where *Ful*, *Com*, *Ctr* and *Can* refer to the sets of fulfilment, commitment, contract and cancellation transactions, respectively.

Trust is a partial order over $\{Ful \cup Com \cup Ctr\}$ X $\{Ful \cup Com \cup Ctr\}$.

Flow is a partial order over Ful X Ful.

A set of rules that govern the orchestration of activities (as defined of a pair of Requesting/Responding Business Activities in a Transaction) can now be defined.

Rule1: If A and B are nodes in a choreography C, and $<A,B> \in \{Flow \cup Trust\}$ then there must exist a path from A to B in C.

Furthermore, we observe that the establishment of a commitment or contract must precede the cancellation of the same, which gives rise to the following rule:

Rule 2: If A and B are nodes in a choreography C and $A \in \{Com \cup Ctr\}$ and $B \in Can$ where B is cancelling the contract or commitment established by A, then there must exist a path from A to B in C [4].

Returning to the relationships between EconomicCommitment, EconomicContract and EconomicEvent, as stated in [17], we observe that Economic Contracts are subtypes of Agreements carrying Economic Commitments that some actual economic exchange will be fulfilled in the future. Thus, we identify the following rule:

Rule 3: If A and B are nodes in a choreography C and $A \in \{Com \cup Ctr\}$ and $B \in Ful$, where B is establishing the economic event that fulfils the commitment established by A, then there must exist a path from A to B in C.

[4] A fulfilment transaction can be performed or not performed but it cannot be cancelled.

6.2 Inter-collaboration Sequencing

Sequencing of transactions make up business collaborations, which in their most basic form are expressed as state charts over transactions. We now turn to the sequencing within the next layer, i.e. between the business collaborations.

We introduce two new partial orders, analogous to the previously defined Trust and Flow. The signatures of the partial orders are given below where *FulC*, *ConP*, *ConO* and *BusiNeg* refer to the sets of Fulfilment-, ContractProposal-, ContractOffer- and Business Negotiation collaborations as defined above.

Trust^C is a partial order over
$\{\text{FulC} \cup \text{ConP} \cup \text{ConO} \cup \text{BusiNeg}\}$ X $\{\text{FulC} \cup \text{ConP} \cup \text{ConO} \cup \text{BusiNeg}\}$.

Flow^C is a partial order over FulC X FulC.

Rule 4 below is analogous to rule 1 and states that trust and flow dependencies should be respected. Rule 5 expresses that fulfilments always follow upon negotiations. Rule 6 states that in a business negotiation, a contract proposal comes before a contract offer. Finally, rule 7 states that transactions and collaborations that do not change social/institutional relationships can be placed anywhere in a larger collaboration.

Rule 4: If A and B are nodes in a choreography C, and $\langle A,B\rangle \in \{\text{Trust}^C \cup \text{Flow}^C\}$, then there should exist a path from A to B in C, i.e. when a trust or flow dependency holds between the two collaborations A and B, A must precede B.

Rule 5: If A and B are nodes in a directed graph C, A ∈ BusiNeg and B ∈ FulC, where B is establishing the economic event(s) that fulfils the commitment(s) within a given contract established by A, then there must exist a path from A to B in C.

Rule 6: If A and B are nodes in a choreography C, A ∈ ConP and B ∈ ConO, where A is the non legally binding negotiation about the terms to be established in B, then there must exist a path from A to B in C.

Rule 7: If a transaction or collaboration pattern carries only pragmatic actions of type <_, None, _>, i.e. where the 'Action' part of the pragmatic action triplet is equal to 'None', then these transactions and collaborations can be included optionally in any collaboration irrespective of order.

Rules 1 - 7 can be used to guide and restrict the design of a choreography, i.e. give suggestions for possible paths between different transactions and collaborations, as well as sequences of these, and rule out incorrect paths.

7 Concluding Remarks

Integrating process and business models poses a number of problems along several dimensions. Differences in focus, abstraction level and domain give rise to different types of discrepancies that must be resolved. Process models may be seen as describing the communicative world, in particular how agents establish and fulfil obligations, while business models depict the social/institutional world where economic relationships such as 'ownership' hold and actions such as transfer of economic resources occur.

The main contribution of this paper is a unified framework to facilitate the integration of business models and process models in e-Commerce. The approach suggested bridges the gap between the communicative aspects of a process model and the social/institutional aspects of a business model. A key assumption of this approach is that an enterprise can be viewed as a set of co-operating agents that establish, modify, cancel and fulfil commitments and contracts. In carrying out these activities, agents rely on so-called pragmatic acts (speech acts), which are actions that change the universe of discourse when a speaker utters them and a recipient grasps them.

Besides facilitating process and business model integration, the proposed framework offers two main benefits:

Simplified Analysis and Design. It will be easier for business users to participate in analysis and design if they are able to express themselves using concepts that have a business meaning (like propose, declare, commit, cancel) instead of using technical concepts like message structures and state machines. Furthermore, the specification of a pragmatic action is simple, as it can be viewed as filling in a template.

Technology Independence. An approach based on pragmatic actions makes it possible to abstract business semantic conversations out of technical messaging protocols, so that pragmatic actions can be used with any technical collaboration protocol (UMM BCP [2], ebXML BPSS[9], BPEL4WS [3], etc). Thus, pragmatic actions provide a clean interface to collaboration protocols.

References

1. "eBTWG – Business Collaboration Patterns/Business Commitment Patterns Technical Specification (BCP^2) – "Monitored Commitments"", accessed 20030401, http://www.collaborativedomain.com/standards/.
2. "UN/CEFACT Modelling Methodology (UMM-N090 Revision 10), accessed 20030328, http://webster.disa.org/cefact-groups/tmg/doc_bpwg.html.
3. Andrews Y., et al., "Business Process Execution Language for Web Services", accessed 20030404, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.
4. Angelov S., Grefen P., "A Conceptual Framework for B2B Electronic Contracting", in *Proc of 3rd IFIP Working Conf.e on Infrastructures for Virtual Enterprises*, 2002.
5. Bergholtz M., et al., "Reconciling Physical, Communicative and Social/Institutional Domains in Agent Oriented Information Systems - a Unified Framework", *Int. WorkShop on Agent Oriented Information Systems*, ER 2003, Springer-Verlag, LNCS 2814.
6. Bergholtz M., et al., "Business and Process Models – a Unified Framework", in *Proc. of eCOMO'2002, 21th Int. Conference on Conceptual Modeling (ER'2002),* Springer-Verlag.
7. Dietz J., "Modelling Communication in Organisations", in *Linguistic Instruments in Knowledge Engineering*, Ed. R.v.d.Riet, pp 131-142, Elsevier Science Publishers, 1992
8. Dignum F. and Weigand H., "Modelling Communication between Cooperative Systems", in *Proc. of the 7th Conference of Computer Advanced Information System Engineering (CaiSE)*, LNCS, Springer Verlag, 1995
9. ebXML Deliverables, *Electronic Business eXtensible Mark-up Language (ebXML),*Valid on 20030328, http://www.ebxml.org/specs/index.htm
10. Gamma E., Helm R., Johnson R., and Vlissides J., "*Design Patterns: Elements of Reusable Object-Oriented Software*", Addison Wesley. 1995
11. Gordijn J., Akkermans J. M. and Vliet J. C., "Business Modelling, is not Process Modelling", *Int. Workshops on Conceptual Modeling Approaches for E-Business and The World Wide Web and Conceptual Modeling*, Springer-Verlag, LNCS 1921, 2000.

12. Jayaweera P., Johannesson P., Wohed P., "Collaborative Process Patterns for e-Business", *ACM SIGGROUP Bulletin 2001/Vol 22, No. 2*
13. Kolp M., Giorgini P., Mylopoulos J., "Information Systems Development through Social Structures", Int. Conf. on Software Engineering and Knowledge Engineering, 2002.
14. Larman C., "*Applying UML and patterns: an introduction to object oriented analysis and design*", ISBN 0-13-74880-7
15. Malone et al.: "Towards a handbook of organisational processes", *MIT eBusiness Process Handbook*, Valid on 20030404, http://ccs.mit.edu/21c/mgtsci/index.htm.
16. Martin, J., Odell, J.: "*Object-Oriented Methods. A Foundation*", Prentice Hall 1994
17. McCarthy W. E., "*REA Enterprise Ontology*", accessed 20030404, http://www.msu.edu/user/mccarth4/rea-ontology/.
18. Moore Scott A., "A foundation for flexible automated electronic communication", *Information Systems Research*, 12:1, 2001
19. Searle J. R., "*A taxonomy of illocutionary acts*", K. Gunderson (Ed.), Language, Mind and Knowledge, Minneapolis: University of Minnesota, 1975.
20. Taveter K. and Wagner G., "Agent-Oriented Enterprise Modeling Based on Business Rules", *Proc. of 20th Int. Conf. on Conceptual Modeling*, Springer-Verlag, LNCS, 2001.
21. van den Heuvel W. and Weigand H., "Coordinating Web-Service Enabled Business Transactions with Contracts", In Eder J. and Missikoff M. (eds). Conf. of Information Systems Engineering (CAiSE'03), Springer-Verlag, LNCS 2681, pp. 568-583, 2003.
22. Wagner G., The Agent-Object-Relationship metamodel: towards a unified view of state and behavior. *Information. Systems. 28* (5): 475-504 (2003)
23. Yu L., and Schmid B. F., "A conceptual framework for agent-oriented and role based workflow modelling", In G. Wagner and E.Yu, editors, Proc. of the 1st Int. Workshop on Agent-Oriented Information Systems, 1999

Author Index